Universidad Zaragoza
1542

Proyecto Fin de Carrera

Integración de recursos efímeros en una infraestructura distribuida de computación

Autor

Marcos Molina Ordovás

Directores

Sergio Hernández de Mesa

Francisco Javier Fabra Caro

Grupo de Integración de Sistemas Distribuidos y Heterogéneos (GIDHE)

Departamento de Informática e Ingeniería de Sistemas

Escuela de Ingeniería y Arquitectura

Universidad de Zaragoza

Curso 2013/2014

RESUMEN

Integración de recursos efímeros en una infraestructura distribuida de computación

Actualmente, la computación se ha convertido en un elemento fundamental para el avance científico y la investigación, y resulta frecuente ejecutar simulaciones cada vez más costosas y que requieren de un mayor número de recursos computacionales.

En el ámbito universitario y científico existe una gran cantidad de recursos de computación que no se utilizan durante gran parte del tiempo que permanecen encendidos. Por este motivo, estos recursos, denominados “recursos efímeros”, podrían ser utilizados para otros fines como el de ser aprovechados para ejecutar simulaciones científicas.

Este Proyecto Final de Carrera tiene como objetivo principal la implementación de una plataforma "middleware" para gestionar dichos recursos efímeros, en un entorno general que se pueda exportar a diversas organizaciones. Asimismo, en el proyecto se plantea la integración de dicho “middleware” con una plataforma de computación que integra diferentes recursos de computación heterogéneos (grid, cluster, cloud, etc.) previamente desarrollada por el Grupo de Integración de Sistemas Distribuidos y Heterogéneos (GIDHE).

Para ello, primero se presenta un estudio de los diferentes tipos y paradigmas de computación existentes hoy en día, con sus plataformas más importantes, seleccionando la plataforma que se ha considerado más adecuada, HTCondor. A continuación, se analiza el entorno general de un escenario de gestión de recursos efímeros y, posteriormente, el entorno correspondiente al Departamento de Informática e Ingeniería de Sistemas (DIIS) de la Universidad de Zaragoza. Utilizando ambas perspectivas, se simula el entorno de los laboratorios de prácticas del DIIS en el entorno de computación en la nube de Amazon. Finalmente, la plataforma se incorpora a la infraestructura del grupo GIDHE, con el objetivo de potenciar su dimensión computacional. Para realizar dicha integración, se ha añadido un componente planificador que selecciona el recurso más apropiado basándose en la ocupación esperada de los mismos.

Además, se ha utilizado un entorno de computación en la nube para simular un entorno de pruebas real. Este enfoque permite sobreponerse a algunos de los problemas derivados de actuar sobre un entorno real como es la falta de permisos de administración o los problemas de acceso físico a los recursos. El uso de una plataforma cloud nos proporciona total libertad de actuación sobre el entorno virtual para probar las características a implementar sin riesgo de ningún tipo sobre el entorno real, y nos permite ahorrar tiempo en el futuro despliegue en el escenario final, habiendo configurado y probado previamente el sistema.

Finalmente, se ha evaluado la utilidad del planificador desarrollado e integrado en la plataforma de computación del GIDHE con respecto a la versión original de la misma Esto ha dado como resultado que la definición de un algoritmo de planificación inteligente ayuda a una mejor utilización de los recursos disponibles.

Quien la sigue, la consigue.

Agradecimientos

A Sergio Hernández, por su entera disposición a ayudar y su infinita paciencia.

A Javier Fabra, por todo lo que he aprendido gracias a él.

A mis padres, por todo el apoyo que me han brindado ahora y siempre.

A los amigos que siempre han estado ahí, en los buenos y en los malos momentos.

Al resto de la gente que de una manera u otra me ha ayudado a llegar hasta aquí.

Índice

Índice de figuras

Índice de tablas

Capítulo 1 – Introducción

Este proyecto ha sido realizado en la Universidad de Zaragoza, en el Departamento de Informática e Ingeniería de Sistemas (DIIS) [W1], con el Grupo de Integración de Sistemas Distribuidos y Heterogéneos (GIDHE) [W2].

En este primer capítulo se presenta la motivación del proyecto, junto con algunos conceptos necesarios para comprender su contenido. Después se enumeran los objetivos que perseguimos con la realización del proyecto, y por último se introduce la estructura de esta memoria y se explican los contenidos de cada uno de los capítulos siguientes.

1.1. Motivación

La informática es una herramienta fundamental para la ciencia y sus experimentos. El problema es que dichos experimentos requieren con mucha frecuencia una gran cantidad de recursos, muchas veces, insuficientes [B1]. Y a medida que van pasando los años y la informática va evolucionando, las necesidades también lo hacen.

Anteriormente, cuando se necesitaba una gran potencia y complejidad de cálculo, se recurría al uso de supercomputadores o pequeños clústeres de computación. Sin embargo, un supercomputador es muy costoso de mantener y utilizar, y no siempre está disponible la posibilidad de utilizar clústeres de computación, por la infraestructura que requieren.

Ha sido en los últimos años cuando han ido apareciendo diversas plataformas capaces de aprovechar los recursos de los crecientes ordenadores personales, y crear de esta forma un equivalente al supercomputador. De esta forma, la creciente cantidad de recursos efímeros disponibles, podrían ser usados para crear plataformas de computación con muy bajo coste, gracias a los nuevos paradigmas *cluster*, *grid* y *cloud* [B3].

En la actualidad la computación científica ha ganado mucho interés. Alguna de sus aplicaciones son las simulaciones numéricas, bien para reconstruir y comprender eventos conocidos (terremotos, erupciones volcánicas, etc.) o para predecir eventos futuros (tiempo atmosférico, etc.); el análisis de datos, mediante ecuaciones (exploración geofísica de petróleo, etc.) o mediante teoría de grafos (modelar redes sociales, sitios web, etc.); etc.

Estos problemas de grandes necesidades de cálculo, requieren:

i) infraestructuras de computación con unos requisitos muy exigentes

ii) paradigmas que faciliten la utilización de dichas infraestructuras.

Como unión de los requisitos, surge primero la computación *cluster*, y después la computación *grid* y la computación *cloud*. Cada uno de estos enfoques tiene un paradigma concreto de desarrollo y despliegue de aplicaciones, pero en esencia son muy similares. Tratan de proporcionar potencia de cálculo a los usuarios.

La computación *cluster* consiste en la unión de varios computadores, con un único fin, trabajar conjuntamente como un solo supercomputador. La computación *grid* tiene el mismo fin, pero

añadiendo la posibilidad de que los recursos estén geográficamente distribuidos, además de poder ser dedicados o no. La computación *cloud* aprovecha los recursos creando una “nube” de servicios, que se ofrecen de forma flexible bajo demanda, utilizando técnicas de virtualización.

En el ámbito universitario en general existe una gran cantidad de recursos de computación (ordenadores ubicados en despachos, laboratorios de docencia e investigación, salas de lectura, etc.) que no son utilizados de manera continua y, por tanto, en determinados momentos del día pueden estar “disponibles” para ser utilizados con otro propósito del habitual, como por ejemplo, con los paradigmas de computación descritos arriba. Estos recursos reciben el nombre de "recursos efímeros".

En particular, el Departamento de Informática e Ingeniería de Sistemas de la Universidad de Zaragoza dispone de una gran cantidad de ordenadores, organizados en varias salas de práctica y laboratorios de investigación, que la mayor parte de su tiempo están infrautilizados. Muchos se quedan encendidos, de manera ociosa. Tenemos pues unos recursos nada despreciables que podríamos utilizar, como mínimo, en su tiempo ocioso. Además, en los tiempos actuales de crisis sería más que interesante poder aprovecharlos, proporcionando a los investigadores una buena capacidad de cálculo sin coste alguno.

El hecho de que exista una gran cantidad de plataformas para gestionar la potencia de cálculo utilizando recursos efímeros, además del almacenamiento y diversos recursos no utilizados, nos da muchas posibilidades para actuar. Sin embargo, también nos supone que en primer lugar hay que realizar un estudio exhaustivo, sobre qué necesitamos realmente y qué plataforma se puede adecuar más a nuestro objetivo. En este proyecto buscaremos una solución general que pueda adaptarse en cualquier entorno, con la posibilidad de una futura ampliación.

Adicionalmente, una de las líneas principales de investigación del Grupo de Integración de Sistemas Distribuidos y Heterogéneos (GIDHE), al que pertenecen los dos directores del proyecto, consiste en la creación de una plataforma de computación, que integra herramientas *cloud* y *grid* [B2]. Dichas herramientas están distribuidas en diversas infraestructuras conocidas, como por ejemplo AraGrid [W3], el clúster Hermes del I3A [W4] o Amazon EC2 [W5]. Sería muy beneficioso integrar los recursos del ámbito universitario como salas de práctica o laboratorios de investigación, con la plataforma existente, mediante una componente que los controlara de alguna forma. Con esto lo que conseguiríamos es añadir potencia computacional a la ya existente.

1.2. Objetivos

Este Proyecto Final de Carrera tiene como objetivo el diseño, modelado e implementación de un componente "*middleware*" para gestionar los recursos del ámbito universitario, de forma que sea capaz de localizar aquellos candidatos que se puedan utilizar para resolver problemas científicos complejos con grandes necesidades computacionales. El núcleo de este componente *middleware* es el encargado de la gestión eficiente de recursos efímeros distribuidos en topologías de red heterogéneas.

Concretamente, en este proyecto se plantea una propuesta para la posterior integración de los recursos disponibles en los laboratorios del Departamento de Informática e Ingeniería de Sistemas ubicados en el edificio Ada Byron. En cualquier caso, de cara a plantear una solución

general que pueda ser aplicada en otros escenarios, se estudiarán los requisitos y restricciones de la inclusión de otros recursos de computación. De esta forma, uno de los requisitos que debe cumplir el proyecto es la posibilidad de integrar otros recursos de computación (ordenadores de otros laboratorios, ordenadores ubicados en despachos, etc.) de forma sencilla. Para posibilitar este aspecto, se plantea la utilización de la infraestructura de computación en la nube de Amazon. Mediante su uso, se podrán establecer, testear y validar diferentes configuraciones, escenarios y casos de uso así como la sencillez en el proceso de integración.

En primer lugar se realizará un análisis del estado del arte en cuanto a soluciones de este tipo y tecnologías que pueden ser utilizadas para detectar las carencias y necesidades más importantes en la gestión de recursos efímeros, y se elegirá la alternativa más adecuada. Después, este componente se desplegará en la nube de Amazon. Y finalmente, se integrará en una infraestructura software capaz de integrar entornos de computación heterogéneos (Grid, cluster, cloud, etc.), que previamente ha sido desarrollada por el grupo de investigación de los directores, con el objetivo de potenciar su dimensión computacional. Como resultado, se verificará la aplicabilidad de la solución sobre problemas concretos de naturaleza científica.

1.3. Organización

A continuación se listan todos los capítulos y anexos, cómo se va a organizar la memoria:

- **Capítulo 2 – Estado del arte.** Hacemos un análisis de los tipos y paradigmas de computación que existen en la actualidad.

- **Capítulo 3 – Análisis y diseño de la solución.** Aquí analizamos de manera general el escenario de recursos efímeros, describimos una solución utilizando la plataforma HTCondor, y planteamos el despliegue del entorno de recursos efímeros.

- **Capítulo 4 – Implementación.** Describimos el entorno, tanto el virtual como el real, la implementación de HTCondor en el sistema y su posterior integración con la Infraestructura del grupo GIDHE.

- **Capítulo 5 – Verificación y análisis de la solución.** Una vez implementado el sistema, realizamos las pruebas necesarias, de funcionamiento general, de fallo y pruebas adicionales. Realizaremos también un experimento para comprobar la eficacia del mediador de recursos efímeros de la Infraestructura.

- **Capítulo 6 – Conclusiones.** Realizamos las conclusiones, tanto a nivel técnico como personal, a las que se ha llegado tras la realización del proyecto. Asimismo, se describen posibles futuras líneas de trabajo.

- **Anexo I – Estado del arte extendido.** Una versión extendida del Capítulo 2, en la que analizamos con mayor profundidad las plataformas de computación más importantes.

- **Anexo II – Descripción de HTCondor y decisiones de instalación.** En este anexo describimos brevemente algunas características de la plataforma HTCondor, junto con las decisiones de instalación más importantes, y también explicamos el proceso de instalación y configuración de HTCondor.

- **Anexo III – Despliegue en Amazon EC2.** Aquí exponemos las características del entorno real, comparado con el entorno creado para el despliegue de HTCondor en las máquinas virtuales.

- **Anexo IV – Pruebas de uso.** Aquí describimos los escenarios posibles y las pruebas realizadas en Amazon EC2 para la comprobación del correcto funcionamiento de HTCondor.

- **Anexo V - Manual de usuario.** Explicamos los conceptos básicos necesarios para utilizar la plataforma HTCondor.

- **Anexo VI – Integración con la infraestructura del GIDHE.** Aquí mostramos la integración de la plataforma HTCondor con la infraestructura desarrollada previamente por el grupo de investigación GIDHE.

- **Anexo VII – Gestión del proyecto.** En este anexo describimos la metodología de trabajo utilizada, la gestión del tiempo y el esfuerzo y las herramientas de gestión utilizadas.

- **Glosario.** Listado de términos y abreviaturas utilizadas con frecuencia, junto con su significado.

- **Bibliografía.** Listado de referencias utilizadas, tanto bibliográficas como web.

Capítulo 2 – Estado del arte

Recientemente, debido a la gran demanda de recursos computacionales para el procesamiento de trabajos que requieren cientos o miles de procesadores, se han desarrollado diferentes tipos y paradigmas de computación para aprovechar la potencia de grandes grupos de computadores.

A diferencia de la computación monolítica (la forma tradicional, en un solo computador físico), la computación distribuida es un término bastante ambiguo, con muchos enfoques y paradigmas distintos que lo componen.

Actualmente existen tres grandes tipos distintos de computación distribuida: computación *cluster*, *cloud*, y *grid* [B4]. Cada uno resuelve una necesidad dados unos recursos. Dependiendo de los recursos de los que dispongamos y del problema que queramos resolver, necesitaremos una plataforma u otra. Para ello, necesitaremos hacer un estudio exhaustivo de los tipos de plataformas que podemos encontrarnos hoy en día, y ver cuál es la que resuelve mejor nuestro problema.

En este capítulo vamos a ver los tipos o enfoques de la computación distribuida y los paradigmas más importantes, con el objetivo de compararlos, sus ventajas e inconvenientes, para finalmente quedarnos con el paradigma que mejor nos convenga para éste proyecto.

2.1. Computación *cluster*

La computación *cluster* [B5] es el tipo más sencillo de computación distribuida, y también el más antiguo. Cuando hablamos de un clúster de ordenadores, nos referimos a un conjunto centralizado de ellos trabajando conjuntamente para un mismo fin. Se trata pues de ordenadores con dedicación exclusiva para resolver los problemas que se le encarguen desde un ordenador administrador.

Un clúster puede tratarse de un supercomputador formado por nodos organizados en racks, o por varios ordenadores personales normales y corrientes. En ambos casos, todos los nodos están unidos en una misma red local y tienen un nodo superior, que es el que controla todos los demás nodos, distribuyendo la carga de trabajo entre ellos.

Se trata de una alternativa a un equipo multiprocesador. La principal ventaja de este modelo es la mejora de la relación coste/rendimiento. Para añadir más potencia/almacenamiento/etc., nos basta con añadir más equipos, ya sean racks profesionales u ordenadores personales.

Tiene como desventajas de que los computadores que forman parte del clúster tienen dedicación exclusiva, no se pueden utilizar para otro fin que el de resolver tareas, y que la programación y el mantenimiento de estos sistemas son costosos. Se trata de un sistema *SSI* (*Single System Image*).

El nodo principal es el encargado de suministrar tareas a los nodos de computación, dispuestos en una red local. El nodo principal es accesible a través de internet, mientras que los demás, sólo son accesibles a través del principal. En la figura 1 podemos ver cómo sería un clúster tradicional de ordenadores.

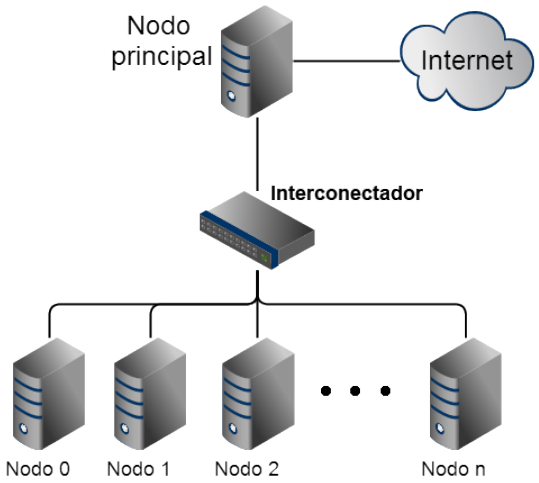


Figura 1: Arquitectura *cluster*

Algunos ejemplos de plataformas de gestión de computación clúster son openMosix [W6], PBS [W7], OpenPBS [W8], LSF [W9] y Grid Engine [W10] (los dos últimos en versión clúster).

2.2. Computación *grid*

La computación *grid* [B6] [B7] es un tipo de computación más abstracto. Su principal objetivo es el mismo que el de la *cluster*, es decir, aprovechar recursos para resolver problemas, de cómputo o de almacenamiento. Algunas claves que cambian en un entorno *grid* respecto a un clúster son la posibilidad de gestionarlo de forma descentralizada, el tipo de recursos que se gestionan, que pueden ser dedicados o no, y el mayor alcance, de mayor dispersión geográfica que un clúster.

La figura 2 muestra un esquema general de cómo podría ser la computación *grid*. La idea general es la de gran cantidad de recursos de todo tipo interconectados entre sí, y distribuidos bien por todo el mundo, bien en una red privada limitada o con un término intermedio.

Figura 2: Arquitectura *grid* [W13]

Es una tecnología que permite utilizar de forma coordinada todo tipo de recursos (computación, almacenamiento y aplicaciones) que no están sujetos a un control centralizado. A diferencia de

la computación *cluster*, con la computación *grid*, los recursos pueden estar distribuidos tanto en una pequeña red local, como en todo internet.

2.2.1. Paradigmas *grid*

Como ya hemos comentado antes, la computación *grid* engloba varios paradigmas. Todos ellos tienen un enfoque común, el de aprovechar los recursos distribuidos. La diferencia está en el alcance de los recursos y en su organización. Hay tres principales paradigmas *grid*:

- ***Intranet computing*.** Similar a la computación en clúster, con la diferencia de que aquí los recursos no son dedicados, por lo que no tienen por qué estar siempre disponibles. Tiene un alcance limitado, normalmente a una, o unas pocas organizaciones. Algunos ejemplos son HTCondor [W12], Grid Engine y LSF.

- ***Internet computing*.** Abarca recursos distribuidos por todo Internet. Aunque la productividad por cada ordenador es bastante más baja, debido a las medidas de seguridad necesarias, se compensa al tener muchas más máquinas disponibles. Cualquiera, donde quiera que esté, pueda donar tiempo de CPU y almacenamiento. Algunos ejemplos son BOINC [W13] y SZTAKI Desktop Grid [W14].

- ***Peer-to-peer computing*.** Similar al paradigma de *Internet computing*, con la principal diferencia de que se trata de un sistema descentralizado, sin un nodo principal [B14]. Se evita así el posible cuello de botella del nodo central, aunque su gestión es más compleja. El ejemplo más representativo es XtremWeb-CH [W14]

2.3. Computación *cloud*

La computación *cloud* [B8] [B9] es un enfoque nuevo basado en la publicación y consumo de servicios y el aprovisionamiento de recursos de computación a través de Internet, bajo demanda y de una forma flexible, adaptable y altamente configurable.

Dentro de la computación *cloud* existen: la Infraestructura como Servicio o IaaS, que proporciona una infraestructura de computación virtual; la Plataforma como Servicio o PaaS, que facilita un servidor de aplicaciones junto con una base de datos; y el Software como Servicio, que ofrece aplicaciones software basadas en la web. En este proyecto nos centraremos en el análisis del paradigma IaaS, que es el centrado en proporcionar recursos de computación para la ejecución de aplicaciones.

La principal diferencia la computación *cloud* de la computación *grid* y *cluster* es que en la primera, el usuario no conoce la ubicación real de los recursos computacionales que está utilizando, de forma que no interactúa directamente con los recursos físicos sino que trabaja en una máquina virtual que puede corresponder a cualquiera de los recursos del proveedor de servicios. Esto posibilita un estilo de computación distribuido, flexible y escalable, gestionado de forma transparente al usuario final [B4].

La figura 3 muestra un ejemplo de arquitectura de un sistema cloud típico.

Figura 3: Esquema general *cloud* [W16]

2.3.1. Tipos de *clouds*

Dentro de la Infraestructura como Servicio o IaaS existen tres tipos de nubes, según su configuración: pública, privada o híbrida [B8].

- **Nube Pública.** Es la que se basa en el modelo estándar de cloud computing en el cual los servicios, aplicaciones y almacenamiento se poden a disposición de los usuarios a través de Internet, normalmente con un modelo de "pague sólo por lo que use".

- **Nube Privada.** Consiste en una infraestructura *cloud* implantada exclusivamente para una única organización, pudiendo ser gestionada interna o externamente. Permite organizar los recursos propios de una organización, sirviendo como alternativa a clústeres o data-centers tradicionales. Las nubes privadas tienen una seguridad avanzada, alta disponibilidad y tolerancia a los fallos que no tienen cabida en la nube pública.

- **Nube Híbrida.** Consiste en una combinación de servicios *cloud* privados (internos) y públicos (externos). Normalmente, las organizaciones ejecutan una aplicación principalmente en la nube privada y utilizan la nube pública de forma excepcional.

Algunos **ejemplos** de nubes públicas son Amazon AWS [W17], Windows Azure [W18] y Google Compute Engine [W19]. Otros ejemplos de nubes privadas e híbridas son Eucalyptus [W20], OpenNebula [W21] y Nimbus [W22].

2.4. Análisis del estado del arte

Este proyecto se va a centrar en plantear un despliegue sobre el ámbito universitario, aunque llegado el caso podría extrapolarse a otras organizaciones similares. Al tratarse de organizaciones limitadas, hemos considerado que el paradigma de computación que mejor se adapta a nuestras necesidades es una herramienta ***grid*** del tipo ***intranet computing***, debido a que son las que mejor rendimiento ofrecen con menor coste, y permiten una gran escalabilidad.

Hemos optado por la computación *grid* debido a las características de los recursos efímeros que queremos aprovechar: sin dedicación exclusiva y con la posibilidad de estar distribuidos geográficamente. Y dentro de *grid*, hemos elegido el paradigma de *intranet computing* al tratarse de organizaciones limitadas, como por ejemplo las universidades.

Las herramientas de *internet computing* tienen el gran inconveniente de la pérdida de rendimiento para contrarrestar la poca seguridad existente en Internet. Y la principal ventaja que tienen, que es la posibilidad de distribuirse por todo el mundo, no la vamos a poder aprovechar. Las herramientas de *cluster computing* nos podrían valer, en el caso de poder configurarlas para que no moleste al usuario/propietario, pero la escalabilidad es menor, además de que no todas las plataformas son open-source o multiplataforma.

De entre las opciones de computación en intranet, hemos elegido HTCondor [B10], por ser open-source, multiplataforma (nos interesa que sea compatible tanto para Windows como para Linux) y tener prácticamente todas las funciones que nos interesan: crear y ejecutar trabajos, una gran escalabilidad, unas políticas de ejecución ampliamente configurables, gestores de trabajos complejos, etc.). También puede adaptarse a otros paradigmas como *cluster* o *internet computing*, aunque con limitaciones.

Además, otro de los objetivos del proyecto es integrar el sistema con una infraestructura de más alto nivel ya existente [B3]. Hemos considerado que con HTCondor, al ser una plataforma completa, más allá que un simple *middleware*, se podría integrar fácilmente con ella.

Para más información, en el Anexo I – Estado del arte extendido se ha hecho un análisis más exhaustivo de los tipos y las plataformas de computación *cluster*, *grid* y *cloud* más importantes, que podrían ser de utilidad en este proyecto.

Capítulo 3 – Análisis y diseño de la solución

En este capítulo plantemos primero análisis del problema: la utilización de los recursos efímeros; después el diseño de su solución, a través de la plataforma HTCondor [B10]; y terminamos planteando el despliegue del entorno de recursos efímeros.

3.1. Análisis del problema

Recordemos que los objetivos del proyecto son el diseño, modelado e implementación de un componente "middleware" para gestionar los recursos efímeros del ámbito universitario.

Para ello, plantearemos una solución general, que podrá ser adoptada en otras circunstancias distintas en las que nos encontramos. Plantearemos primero una visión general de nuestro sistema, y después una serie de requisitos funcionales y no funcionales que debe cumplir.

3.1.1. Visión general del escenario de recursos efímeros

La figura 4 nos da una visión genérica del escenario de recursos efímeros, que puede representar no sólo el ámbito universitario, sino cualquier organización en general. Está compuesto por redes de ordenadores para ejecución (tanto redes públicas como privadas), y una zona de más alto nivel, encargada de gestionar dichos recursos de ejecución.

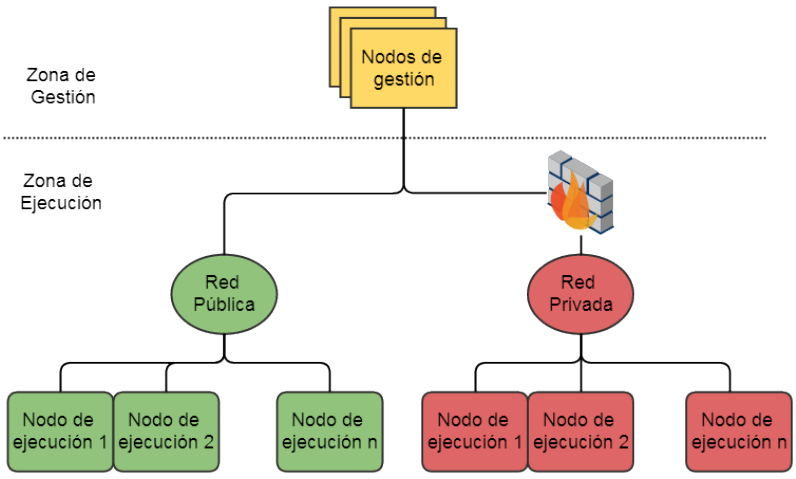


Figura 4: Esquema genérico del sistema

En la figura podemos distinguir dos zonas: la zona de ejecución, distribuida entre varias redes públicas y/o privadas, que es donde se ejecutan los trabajos; y la zona de gestión, encargada de gestionar la zona de ejecución (reparto de trabajos, distribución de carga, monitorización, etc.)

3.1.2. Requisitos funcionales

En este apartado vamos a ver los servicios y comportamientos que debe prestar el sistema. Primero veremos los relativos a las funciones del sistema y luego los que hacen referencia a otros criterios distintos.

RF-1 El sistema debe permitir a los usuarios mandar trabajos desde cualquier ordenador perteneciente al mismo.

RF-2 El ordenador en el que se ejecute una tarea debe cumplir las necesidades de procesador y almacenamiento que le han sido especificadas al mandar la tarea.

RF-3 La plataforma no debe exceder el tamaño de disco reservado para ella, y en cualquier caso, el tamaño debe ser menor que el tamaño disponible del disco físico.

RF-4 La plataforma no debe entorpecer el acceso a los ordenadores de prácticas. Esto significa que cuando un recurso esté siendo utilizado, no pueden ejecutarse tareas en él.

RF-5 El sistema debe ser monitorizable. El administrador debe tener herramientas a su disposición para controlar el estado de los componentes, detectar fallos y restaurar el sistema.

RF-6 El sistema debe ser capaz de realizar *checkpoints* en diversas condiciones.

RF-7 El sistema debe tolerar y gestionar correctamente las posibles caídas frecuentes de los recursos efímeros.

3.1.3. Requisitos no funcionales

RNF-1 El sistema se debe poder instalar a coste cero.

RNF-2 La plataforma debe ser transparente al usuario.

RNF-3 El nodo central debe poder acceder a todos los recursos pertenecientes al sistema, incluso si estos se encuentran tras un firewall o una red privada.

RNF-4 El sistema debe ser escalable. Se deben poder añadir o eliminar ordenadores sin comprometer el rendimiento. Incluso si éstos se encuentran tras una red privada, a priori no accesible desde el nodo central.

RNF-5 La plataforma instalada se debe poder integrar con la infraestructura desarrollada por el grupo de investigación GIDHE [W2].

3.2. Diseño de la solución

Como ya hemos visto en el capítulo anterior, la plataforma que hemos elegido para la gestión de los recursos efímeros es HTCondor, porque es la que mejor se adapta a nuestro entorno y a nuestras necesidades.

En este apartado describimos las características más importantes de HTCondor. También planteamos varios esquemas posibles de configuración, así como diversas alternativas de configuración con sus ventajas e inconvenientes, y algunas consideraciones de seguridad.

3.2.1. Descripción de HTCondor

HTCondor es un sistema formado por diferentes componentes y funciones. Las máquinas con HTCondor pueden tener 3 funciones principales distintas: ***Central Manager***, ***Submit*** y ***Execute***. Cada ordenador puede tener una o más funciones, según su configuración. Cada función corresponde a uno o más *demonios*.

El ***Central Manager*** es el ordenador principal, que controla los demás recursos. Es el encargado de unir peticiones con recursos disponibles (*negociador*) y de recopilar información sobre qué equipos están disponibles, cuáles dejan de estarlo, etc. (*colector*).

Las máquinas ***Submit*** son las que tienen capacidad para someter trabajos que otras ejecutarán. Las ***Execute*** son la esencia de la plataforma HTCondor, las que ejecutan los trabajos que les mandan las máquinas *Submit*.

Existen además diversas funciones auxiliares. Entre ellas, nos interesa la llamada ***Condor Connection Broker*** (*CCB*) cuyas tareas son dos: actuar como *proxy* entre todas las comunicaciones entrantes y salientes de un recurso, evitando el número de puertos de escucha necesarios a uno sólo, el 9618; y permitir al *colector* registrar recursos privados para poder acceder a ellos normalmente. Por tanto, de no estar configurado un *CCB*, no es posible el acceso a estos recursos privados.

Antes de enumerar las distintas alternativas, debemos considerar varios supuestos que debemos respetar en cualquier caso.

- El más importante es que el *Central Manager* tiene que estar en una máquina estable, que no esté sometida a caídas frecuentes. Al ser el que controla todas las demás máquinas, si éste cae, todo el sistema se vuelve no funcional.

- Debemos tener permisos para administrar el *Central Manager*, para poder arreglar cualquier fallo que pudiera ocurrir en el menor tiempo posible.

- Es necesario que el *Submit* esté en una máquina estable. Se permitiría sin embargo que un *Submit* estuviese en una máquina inestable si fuera un espejo del *Submit* principal, para evitar un problema mayor.

En el apartado II.1. Descripción de HTCondor describimos la plataforma en mayor profundidad y con más detalles técnicos.

3.2.2. Análisis de alternativas

Una vez que hemos visto el esquema general del sistema en la figura 4, y las funciones y necesidades principales de la plataforma HTCondor, en este apartado vamos a analizar las diferentes posibilidades para el despliegue de HTCondor en sistemas de recursos efímeros.

Debido a los supuestos que debemos respetar en cualquier caso (comentados arriba), las alternativas viables se reducen considerablemente. Las máquinas ejecutables se encontrarán siempre en la zona de ejecución. Al mismo tiempo, las máquinas de gestión (nodo principal y herramientas auxiliares) deberán estar en la zona de gestión. Las alternativas aquí propuestas vendrán diferenciadas sobre todo por la ubicación de las máquinas *Submit*.

3.2.2.1. Alternativa centralizada

En este caso el nodo principal hace tanto de *Central Manager* como de *Submit*, y todas las demás son los nodos *Execute*. La figura 5 muestra gráficamente cómo sería esta configuración.

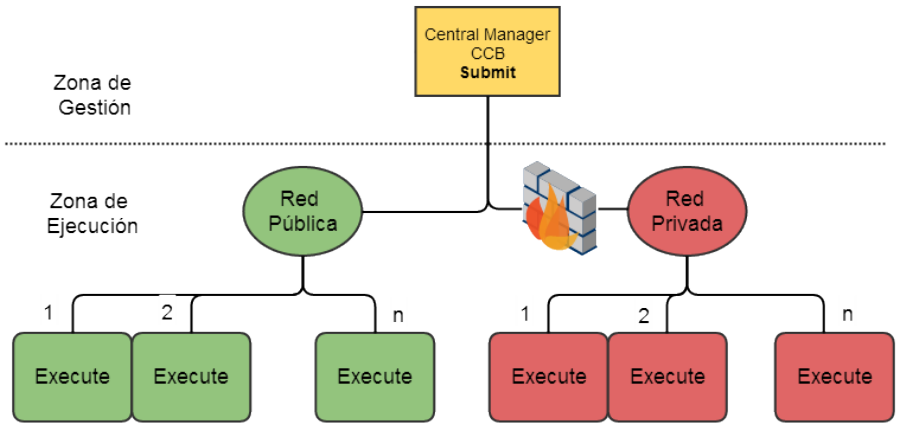


Figura 5: Alternativa de configuración centralizada

Ésta es la alternativa más simple, y la más fácil de configurar. Tiene las zonas de gestión y de ejecución bien diferenciadas. Sin embargo, tiene la desventaja de que si el nodo principal se cae, se paraliza todo el sistema.

3.2.2.2. Alternativa descentralizada

Aquí el nodo principal hace únicamente de *Central Manager*, mientras que todas las demás máquinas son nodos *Submit* y *Execute* simultáneamente. La figura 6 muestra el esquema de la alternativa de configuración.

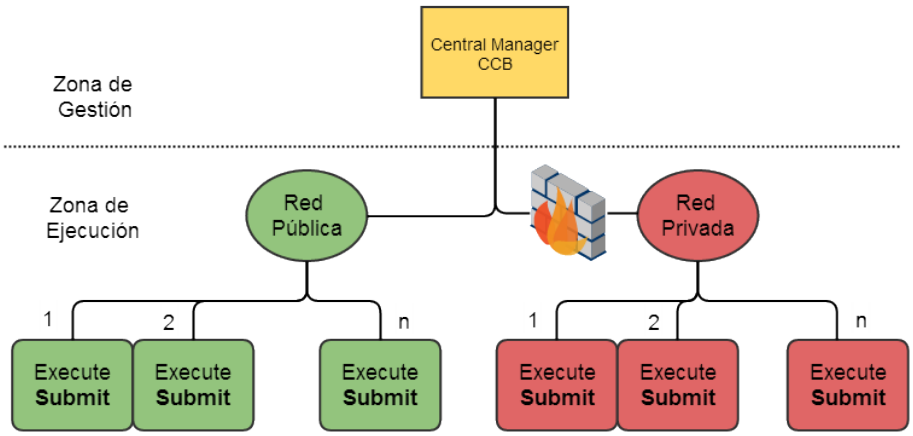


Figura 6: Alternativa de configuración descentralizada

Con este esquema, cualquier máquina (salvo el nodo central) dispone de una cola de trabajos, por lo que podría someter. Da más libertad al usuario, pero también complica mucho su gestión si hay caídas frecuentes de máquinas.

3.2.2.3. Alternativa híbrida

Esta es una solución intermedia. El *Central Manager* tiene el *Submit* principal, pero esta vez configuramos algunas máquinas (preferiblemente con IP pública) como espejos del mismo. La figura 7 describe dicha configuración.

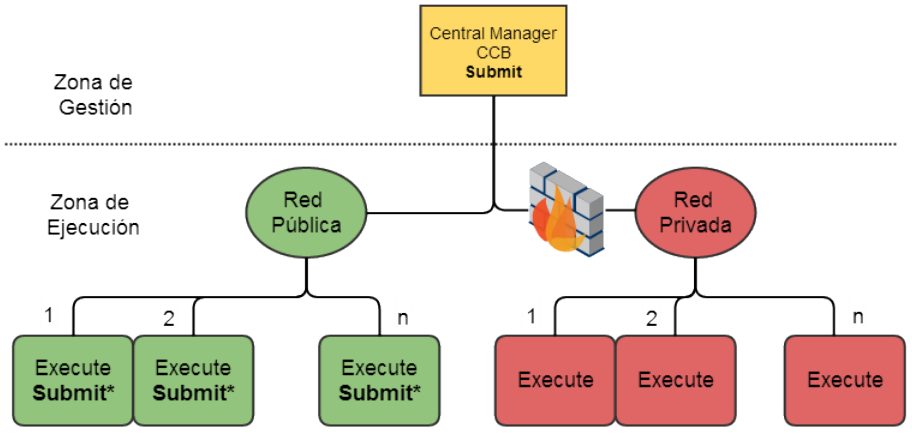


Figura 7: Alternativa de configuración híbrida

El hecho de que existan réplicas de la máquina *Submit* principal minimiza el impacto de una posible caída de éste. Evitaría de este modo que los trabajos sometidos se perdieran, y además permitiría seguir mandando trabajos. Como desventaja principal, la cola de trabajos ha de estar en un directorio compartido por todas las máquinas *Submit*, algo que no siempre es posible.

3.2.2.4. Valoración de las alternativas

La alternativa 1 es la más fácil de configurar, sin embargo tiene varias desventajas. Al configurar en el *Central Manager* todas las funciones excepto la de ejecutar tareas, concentramos en él un punto crítico mayor de lo que sería deseable. Si tuviera cualquier fallo, se paralizaría todo el sistema. Además del hecho que concentrar todas las funciones puede provocar una sobrecarga en la máquina y que falle algún proceso.

En la alternativa 2 es una solución descentralizada. En este caso, el Central Manager únicamente realiza las funciones de *negociador* y *colector* (y si es necesario, de servidor *CCB*). Esto tiene un problema con la función *Submit*, y es que el hecho de que cualquiera pueda someter, y a la vez se trate de recursos que en cualquier momento pueden dejar de estar disponibles, puede hacer muy difícil la gestión de los trabajos. Cada vez que una máquina *Submit* deja de estar disponible, todos los trabajos en ejecución que se han sometido desde esa máquina se pierden y habría que volver a someterlos desde otra máquina. Esto último es especialmente grave y nos hace imposible adoptar esta configuración.

La alternativa 3 corresponde a una solución intermedia. Por un lado, reducimos el impacto que podría ocasionar la caída del *Central Manager*, creando espejos ("*mirrors*") del *scheduler* principal (el proceso que controla la cola de trabajos) en algunas máquinas con IP pública. De esta forma, si el principal cayera, no se paralizaría todo el sistema, puesto que se levantaría cualquiera de sus espejos, y tampoco se perderían los trabajos sometidos, dato importante. Esta alternativa requiere que todos los *schedulers*, tanto el principal como sus espejos, dispongan de una misma carpeta compartida. Conviene que estén en máquinas con IP pública, para que sean accesibles para todos.

Dado que en nuestro caso no ha sido posible la implementación de una carpeta compartida entre todos los nodos *Submit*, nos hemos decantado por la **alternativa centralizada**. Sin embargo, en un futuro sería deseable implementar la alternativa híbrida de ser posible configurar una carpeta compartida, por las ventajas que tiene. Y en el ámbito universitario o en organizaciones similares, se suele dar ese caso, y los cambios adicionales a realizar entre las dos alternativas son mínimos.

En nuestro caso, para reducir la limitación que supone tener un único punto para someter trabajos, vamos a habilitar la función de someter remotamente. Se trata de una función auxiliar que nos proporciona HTCondor, la cual nos permitirá someter trabajos sin conectarnos directamente al *Central Manager* (donde se encuentra el *scheduler*) y evitar la sobrecarga que supone el acceso concurrente de varios usuarios a la misma máquina. Se podrá de esta forma, someter desde cualquier nodo del sistema. Como inconveniente, HTCondor nos obliga en este caso a implementar como mínimo, un método de autenticación de usuarios. En el siguiente apartado analizamos los diferentes métodos de autenticación existentes, que nos permitirán proporcionar la seguridad requerida por HTCondor.

3.2.3. Consideraciones de seguridad

Tal y como hemos visto en el apartado anterior, hemos propuesto implementar la función de someter trabajos remotamente. Aunque para ello, HTCondor nos obliga a autenticar a los usuarios del sistema, por seguridad. Tendremos que implementar como mínimo, algún sistema de autenticación que permita identificar los diferentes usuarios.

En este apartado vamos a analizar las alternativas que nos proporciona HTCondor únicamente para la autenticación. HTCondor también nos da la posibilidad de implementar funciones de encriptación e integridad, pero como el entorno en el que trabajamos tiene tanto máquinas como usuarios confiables, no serán necesarias.

Existen varios métodos de autenticación soportados por HTCondor. Sin embargo, no todos son multiplataforma, ni todos son igual de seguros o complejos de configurar, ni sirven para lo mismo. En la tabla 1 vemos un resumen de las características de los distintos métodos [W23]. En ella mostramos las plataformas compatibles para cada método, el nivel de seguridad que proporcionan (según la capacidad que tienen para evitar intrusiones y situaciones similares, no deseadas), su complejidad a la hora de desplegarlas (evaluado en cuanto a la dificultad para desplegar el método) y los propósitos para los que se pueden utilizar.

Métodos	Plataformas	Seguridad	Complejidad	Propósitos
FS	Linux	Alta	Baja	Sólo en local
FS Remote	Linux	Variable	Media	Cualquiera con acceso al NFS
Password	Cualquiera	Alta	Media	Sólo entre demonios
GSI	Linux	Alta	Muy alta	Cualquiera
SSL	Cualquiera	Alta	Alta	Cualquiera
Kerberos	Cualquiera	Alta	Muy alta	Cualquiera
NTSSPI	Windows	Baja	Baja	Cualquiera, sólo autenticación
Claimtobe	Cualquiera	Ninguna	Baja	Debug
Anonymous	Cualquiera	Ninguna	Baja	Debug

Tabla 1: Alternativas de seguridad

Tras analizar los resultados, los únicos métodos que permiten someter remotamente y son multiplataforma son SSL [W24] y Kerberos [W25]. Hemos decidido utilizar **SSL** por ser menos complejo y más conocido.

En el apartado II.2.1. Consideraciones de seguridad explicamos con mayor detalle todos los métodos, y en el apartado II.3.7. Seguridad y autenticación, la configuración y el despliegue de la parte de seguridad.

3.3. Despliegue del entorno

Una vez hemos definido la configuración en un entorno general, procederemos a desplegar el sistema para poder configurarlo. Esto lleva consigo varios problemas: problemas de acceso, de administración, disponibilidad limitada, etc. Para solventar dichos problemas, una buena solución consiste en utilizar un entorno virtual a través de Amazon EC2 [W5]. También se podría haber hecho con otros proveedores *cloud* como Google Compute Engine [W19] o

Rackspace [W26], pero en su momento consideramos que la mejor opción era la de Amazon EC2.

El hecho de utilizar un entorno virtual como el que nos proporciona Amazon EC2 para simular un entorno real presenta múltiples ventajas tanto en la referente a la gestión de los propios recursos como a la del software de gestión de recursos efímeros utilizado HTCondor. Esto nos permite probar, eliminar máquinas para volver a crearlas, etc. En general, facilita la administración del sistema, su configuración, la comprobación de errores, etc.

De forma más específica, el uso de una plataforma *cloud* para simular un entorno de recursos efímeros ofrece ventajas sobre la gestión de los recursos (máquinas virtuales) y la administración del entorno. A continuación, enumeramos las ventajas más importantes.

- Nos permite crear configuraciones de red de forma sencilla, con la posibilidad de cambiarlas o eliminarlas en cualquier momento sin esfuerzo.
- Podemos probar situaciones o configuraciones que no podríamos realizar en el entorno real, ya sea por problemas de administración, falta de permisos o falta de recursos.
- Nos proporciona mayor facilidad para simular situaciones de error, y diseñar y evaluar técnicas de recuperación de los errores.
- En general, nos ahorra mucho tiempo y esfuerzo si lo comparamos con el despliegue en máquinas reales.
- Tenemos acceso a todas las máquinas en cualquier momento.
- Una característica muy útil de Amazon es que cada máquina virtual se puede identificar con una IP privada y con otra IP pública, ambas son únicas e identifican a la misma máquina, con la diferencia que la privada sólo es accesible desde dentro de la red VPC mientras que la pública no tiene limitaciones en ese sentido. Gracias a esto, podemos suponer que las máquinas tienen IP pública o privada simplemente utilizando una u otra dirección. Esto simplifica la gestión y facilita el testeo del sistema bajo diferentes condiciones.

Respecto a las ventajas relativas a la administración, instalación y configuración de las máquinas virtuales, podemos destacar las siguientes:

- Disponemos de permisos de administración en todas las máquinas, lo que nos evita todo el proceso necesario para poder hacer cambios en el entorno real.
- Uno de los servicios más importantes que Amazon EC2 proporciona es la posibilidad de poder crear copias exactas del disco duro de una máquina virtual, para poder lanzar máquinas clones de las mismas. Esto nos facilita la realización de copias de seguridad y evita tener que configurar todas las máquinas una por una.
- Al poder restaurar en cualquier momento una máquina a partir de una copia de seguridad anterior, podemos realizar un despliegue fácilmente en todas las máquinas sin comprometer el resto del sistema y que afecte a otros usuarios.
- Podemos configurar puertos y políticas de seguridad del firewall.

Capítulo 4 – Implementación

En este capítulo veremos detalles más concretos, del nivel de implementación. Comenzaremos viendo el entorno de implementación, tanto real como virtual. Más tarde explicaremos el proceso de instalación y configuración de HTCondor [B10]. Y finalizaremos detallando la integración de nuestro sistema con la infraestructura del grupo GIDHE [W2].

4.1. Descripción del entorno

En el capítulo anterior, hemos visto un planteamiento de despliegue en el ámbito universitario general. En este apartado veremos un planteamiento ya más concreto, referido a los recursos del Departamento de Informática e Ingeniería de Sistemas [W1]. Hemos utilizado este escenario como referencia por ser suficientemente general y representativo, y por la familiaridad que tenemos con el mismo. También crearemos un entorno virtual, como simulación del real.

4.1.1. Descripción de los laboratorios

La figura 8 muestra la topología simplificada de los ordenadores del Departamento de Informática e Ingeniería de Sistemas. También se ha añadido el nodo central, ubicado en el laboratorio del grupo GIDHE en el I3A.

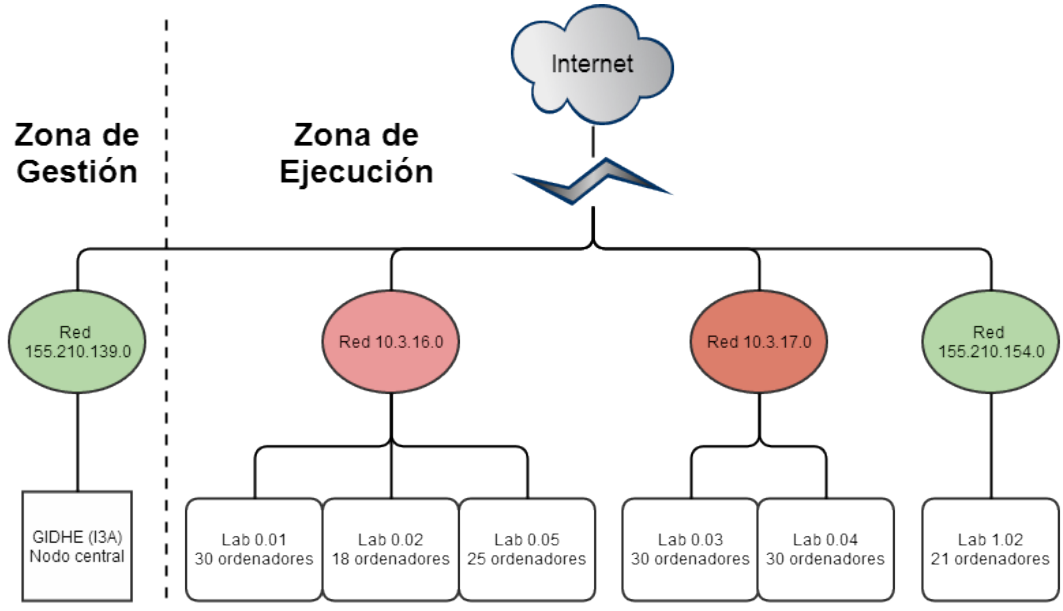


Figura 8: Topología del sistema

Nuestros recursos disponibles inicialmente van a ser los ordenadores de los 6 laboratorios de prácticas. Existe la posibilidad de incorporar nuevos recursos en un futuro, por lo que a la hora de realizar el despliegue, tendremos este hecho en cuenta.

Conceptualmente podemos distinguir tres tipos de máquinas entre todas ellas. Primero tenemos el servidor personal, con IP pública y que además está siempre encendida. Luego tenemos las

máquinas del laboratorio 1.02, con IP pública, que además son las más potentes. Y finalmente, las de los demás laboratorios, con IP privada, sólo accesibles desde dentro de la universidad, incluyendo el nodo principal. De todas ellas, sólo el nodo principal será considerado como "estable", y por tanto pertenecerá a la zona de gestión. Todas las demás máquinas las consideramos como susceptibles a caídas frecuentes, por poder ser utilizadas por alumnos en cualquier momento, y estarán dentro de la zona de ejecución.

- Un ordenador personal en el I3A, con IP pública. Será el único que tengamos con permisos de administrador, y que consideremos "estable", es decir, que no esté sujeto a caídas frecuentes. Lo configuraremos como ***Central Manager***. También será el que controle la cola de trabajos, es decir, el ***Submit***. Y un servidor ***CCB***, para posibilitar la incorporación de máquinas con IP no accesibles desde aquí.

- Los 21 ordenadores del laboratorio 1.02. Éstos son especiales porque tienen IP pública, y por tanto son accesibles desde todo el mundo, además de ser los más potentes. En un principio los plantearemos como máquinas ***Execute***, aunque dejaremos la posibilidad de que en un futuro sean configurados también como **espejos del *Submit*** principal. Esto se debe a que en el DIIS existe la posibilidad de crear una carpeta compartida con otras máquinas del Departamento.

- Los ordenadores de los demás laboratorios, pertenecientes a las redes privadas 10.3.16.0/24 y 10.3.17.0/24. Un total de 133 ordenadores repartidos en 5 laboratorios (en la práctica habrá menos disponibles al mismo tiempo). Todos ellos los plantearemos como máquinas ***Execute*** únicamente.

Las características concretas de cada laboratorio se encuentran definidas con más detalle en el apartado III.2. Características de los laboratorios.

4.1.2. Despliegue en Amazon EC2

Ya hemos visto en el apartado 3.2.2. Alternativas de configuración, las diferentes alternativas de configuración, eligiendo la más adecuada. También acabamos de ver en el apartado anterior la configuración de red de los laboratorios del DIIS. Partiendo de esta información, vamos a crear un entorno simulado en Amazon EC2 lo más parecido posible al entorno real de los laboratorios del DIIS, por ser lo suficientemente general. Esto nos permitirá comprobar su correcto funcionamiento más adelante.

Para simular la configuración de red de los laboratorios, Amazon nos facilita la función VPC (Virtual Private Cloud) [W27], que permite crear redes virtuales privadas, con varias subredes y una puerta de enlace con salida a internet. Con ella, hemos creado una red VPC con salida a Internet, 10.3.0.0/16. Y dentro de ella, tres subredes: 10.3.16.0/24, 10.3.17.0/24 y 10.3.154.0/24, coincidiendo con las subredes de los laboratorios reales.

Para simular las máquinas, utilizaremos instancias de CentOS 6.4 de 64 bits (el sistema operativo de los ordenadores de los laboratorios) y alguna de Windows 2008 R2. Instalamos el paquete HTCondor y otras herramientas necesarias, y creamos una imagen del disco o AMI para el *Central Manager* y otra para las demás máquinas (por ser diferente su configuración). Una vez hecho esto último, podremos crear instancias clones de la imagen de CentOS. Únicamente

cambian entre ellas la configuración de red y el nombre de la máquina (basado en la IP). Podríamos crear también otra para las máquinas con Windows, pero como únicamente utilizaremos una o dos para probar, y la instalación es más sencilla, no va a ser necesario.

Llegados a este punto, creamos varias instancias clones de la que hemos configurado, en las distintas subredes que acabamos de crear. En la red 10.3.154.0 irán el *Central Manager* y las máquinas del laboratorio 1.02. En la red 10.3.16.0 van las de los laboratorios 0.01, 0.02 y 0.05. Y la 10.3.17.0, las de los laboratorios 0.03 y 0.04.

Por último, para simular las restricciones de red y el firewall, hemos creado varios "Grupos de Seguridad", conjuntos de reglas de firewall para controlar el acceso a las máquinas virtuales. Permitiremos el acceso al puerto 22 (SSH) en Linux y 3389 (RDP o escritorio remoto) en Windows, y luego en todas, el puerto 9618 (*CCB* de HTCondor), y los paquetes ICMP para poder hacer ping, para pruebas. El entorno después de todo queda según muestra la figura 9:

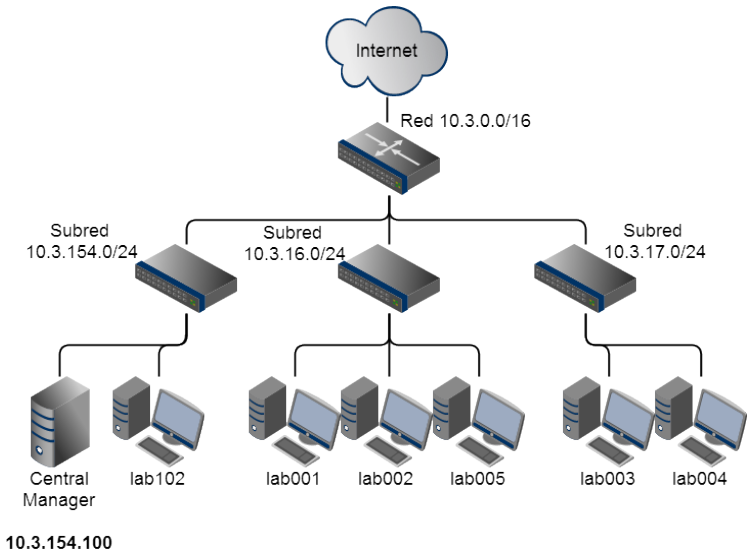


Figura 9: Esquema general de despliegue en Amazon EC2

4.2. Implementación de HTCondor

En el apartado de arriba hemos visto la configuración de red de los laboratorios primero. También hemos comentado el entorno virtual de Amazon EC2 creado para la ocasión, como una simulación de los laboratorios reales.

Una vez conocido el entorno, vamos a proceder a la instalación y configuración de la plataforma HTCondor. Primero detallaremos los pasos a seguir para instalar HTCondor, continuaremos con la configuración básica de la plataforma, y finalizaremos con su configuración más avanzada.

4.2.1. Instalación

En Linux hay tres formas de instalar HTCondor en Linux: compilando el código fuente y distribuyendo los ficheros resultantes; instalando desde un paquete *.deb* (Ubuntu/Debian/etc.) o *.rpm* (CentOS/Red Hat/etc.); e instalando desde repositorio, a través de `apt-get` (deb) o `yum` (rpm). Para hacerlo de una forma sencilla, y a la vez flexible, nosotros instalaremos las dependencias de HTCondor mediante repositorio (`apt-get` o `yum`), y a partir de ahí instalaremos HTCondor desde un paquete *.deb* o *.rpm*.

La opción de compilar la descartamos, porque no necesitamos tocar el código fuente ya que no se van a modificar funcionalidades de HTCondor, la parte importante es la de configuración, que viene después. Se podría instalar todo desde repositorio, pero como los repositorios suelen tener versiones desfasadas o inestables, consideramos que lo más correcto es a través del paquete de la página oficial que nos ofrece la última versión estable.

En Windows sólo hay una forma, que es descargando el instalador *.msi*, ejecutando y siguiendo los pasos. La configuración que se introduzca durante la instalación no es muy importante, pues luego habrá que modificar a mano el fichero de configuración.

4.2.2. Configuración básica

Dentro de lo que se entiende como configuración básica, tenemos la configuración del *pool*, los roles que desempeña cada máquina, la política de ejecución y el servidor *CCB* (*HTCondor Connection Broker*).

La **configuración del *pool*** los parámetros principales que tendrá dicho *pool*. Entre ellos se encuentra la dirección del *Central Manager* (el parámetro más importante), las rutas de los ficheros, la dirección de correo del administrador, el dominio de red y los permisos de acceso.

Los **roles** establecen el papel que desempeñará cada máquina. Puede haber más de uno a la vez.

La **política de ejecución** es lo que define cuándo una máquina está disponible para ejecutar trabajos, cuándo deja de estarlo, qué se hace con los trabajos de una máquina que deja de estar disponible y otros casos similares. El uso más común de dicha variable es el de preferencia al propietario o grupo de propietarios de una máquina, antes que el de otros.

El **servidor *CCB*** es una función adicional del colector, que actúa como registrador. Cada máquina con IP privada no accesible, se configura para que se registre al inicio en el *CCB*. De esta forma, se mantiene siempre una conexión bidireccional y es accesible por el nodo central.

En el apartado II.3. Configuración de HTCondor se muestra toda la configuración con mayor detalle y con fragmentos del fichero de configuración.

4.2.3. Configuración avanzada

La parte de configuración avanzada comprende las funciones de autenticación, de *alta disponibilidad* y de *flocking*.

La **autenticación** permite a HTCondor verificar la identidad de un usuario concreto, a través de un par certificado – clave privada de SSL. Para que un usuario pueda someter un trabajo, será necesaria su autenticación. No es necesario si sólo se pretende ejecutar.

La ***alta disponibilidad*** consiste en la creación de espejos que añaden redundancia al sistema, y por tanto, seguridad. Pueden ser espejos del *Central Manager* y de la cola de trabajos. Dadas sus limitaciones, esta función la planteamos para ser configurada en un futuro, no ahora.

Flocking es una característica de HTCondor para permitir que dos o más *pools* distintos se junten a través de su colector, compartiendo y complementando los recursos disponibles. Para ello, es necesario que cada colector tenga las direcciones de todos los demás, junto con los permisos necesarios.

El apartado II.3. Configuración de HTCondor contiene toda la configuración con mayor nivel de detalle y con fragmentos del fichero de configuración.

4.3. Integración con la infraestructura del GIDHE

Tal y como hemos mencionado anteriormente, uno de los objetivos de este proyecto es el de integrar nuestra plataforma HTCondor desplegada en el ámbito universitario, con una infraestructura desarrollada previamente.

En este capítulo describiremos dicho proceso de integración, comenzando por la visión general de la infraestructura, siguiendo por los cambios realizados, y finalizando con el planteamiento de varias líneas desarrollo futuro.

4.3.1. Descripción de la infraestructura

El grupo de investigación GIDHE [W2] ha desarrollado una infraestructura orientada a servicios para el despliegue y la ejecución de *workflows* científicos en entorno de computación heterogéneos. Esta infraestructura integra y gestiona de forma transparente un amplio conjunto de recursos de computación (recursos tipo cluster, grid, cloud, etc.), y los ofrece al usuario como un único y potente entorno de ejecución.

La figura 10 muestra el diseño arquitectural de la infraestructura el cual está formado por tres capas. En la parte superior, la parte de interfaz de usuario permite programar las aplicaciones a ejecutar utilizando diferentes lenguajes. En la parte intermedia, la capa de ejecución es la responsable de gestionar el despliegue de las aplicaciones en las infraestructuras de computación integradas. Para ello, incluye dos tipos de componentes que se comuniquen a través de un bus de mensajes: los componentes de gestión que gestionan el ciclo de vida de las aplicaciones y los mediadores que interactúan con cada infraestructura concreta a través de su

middleware. Finalmente, en la parte inferior, la capa de infraestructuras engloba los recursos utilizados para ejecutar las aplicaciones desplegadas.

Puede consultarse más información sobre la infraestructura de computación del GIDHE en [B2] [B3].

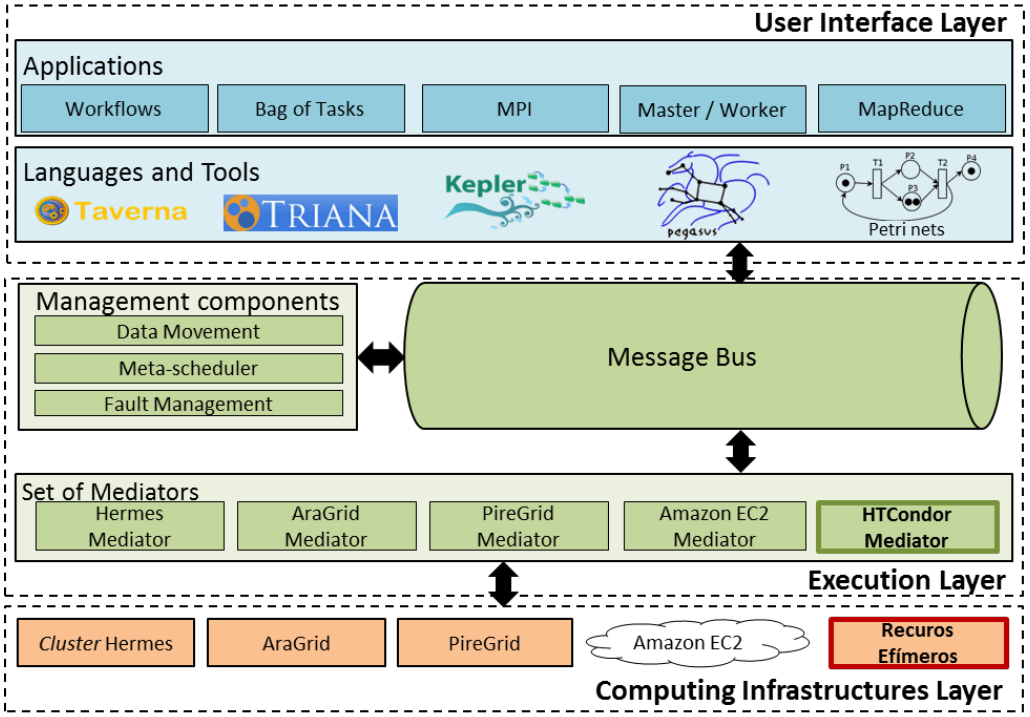


Figura 10: Arquitectura de la infraestructura

Una de las partes más importantes de la infraestructura, y la que más atañe a este proyecto, son los mediadores englobados en la capa de ejecución. El diseño de los mismos puede observarse en la figura 11. Este diseño es común para todos los mediadores, independientemente de la infraestructura y el middleware con el que interactúen, mientras que la implementación de los mismos varía según estos dos aspectos. En lo que respecta a los componentes que forman el mediador:

El **Job Manager** se encarga de gestionar y coordinar la ejecución de los diferentes trabajos enviados al mediador. Para ello interactúa con el **Bus** de mensajes para obtener nuevos trabajos y enviar sus resultados, gestiona el movimiento de los datos generados como resultado a la ubicación adecuada y mantiene una lista de los trabajos en ejecución junto con los identificadores asignados. El **Middleware Adapter** es el encargado de traducir la descripción del trabajo sometido, la cual es independiente del entorno de ejecución, en una descripción que pueda ser ejecutada por el middleware específico con el que interactúa. Asimismo, prepara los datos necesarios para poder ejecutar el trabajo. El **Middleware Executor** obtiene la descripción del trabajo a ejecutar y envía el mismo al middleware para que éste proceda a su ejecución. Finalmente, el **Job End Monitor** se encarga de detectar cuando un trabajo ha terminado y notificar al Job Manager.

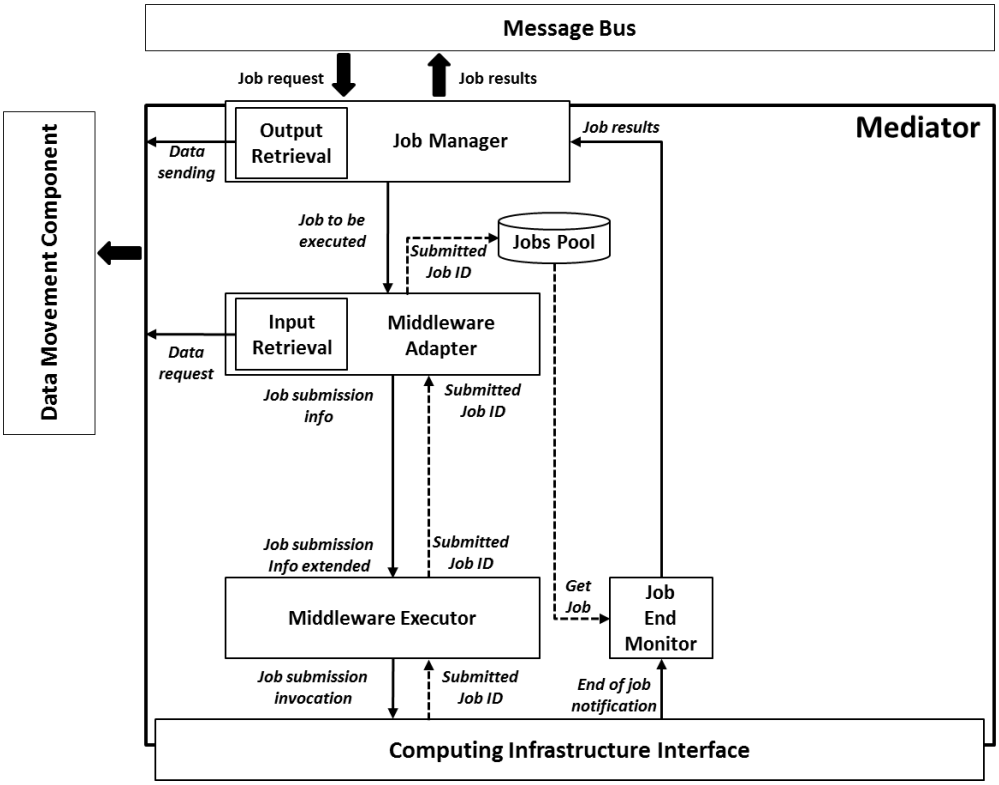


Figura 11: Esquema del mediador original

4.3.2. Funcionalidades añadidas

En este proyecto se ha extendido la infraestructura presentada anteriormente para que sea capaz de integrar entornos de computación compuestos por recursos efímeros de ámbito académico. Más concretamente, se ha extendido el diseño de los mediadores de la infraestructura para que incluyan nuevas funcionalidades encaminadas a gestionar este tipo de entornos. La inclusión de estas nuevas características resuelve el problema de que los mediadores estaban diseñados para ser utilizados en entornos de computación estables en los que los recursos no pueden ser añadidos o eliminados de forma dinámica por causas externas y sin previo aviso (salvo aparición de errores). Por tanto, mediante el desarrollo de este nuevo mediador se consigue realizar un mejor aprovechamiento de los recursos integrados.

Una característica extra de HTCondor que nos va a ser muy útil es la que permite establecer máquinas o grupos de máquinas prioritarios para la ejecución, mediante la variable RANK. Es decir, que si hay capacidad de decisión en el momento de elegir máquina para ejecutar, se elegirá antes a una prioritaria.

La figura 12 muestra el diseño extendido del mediador, remarcando en color los nuevos componentes añadidos al mismo, los cuales se detallan a continuación:

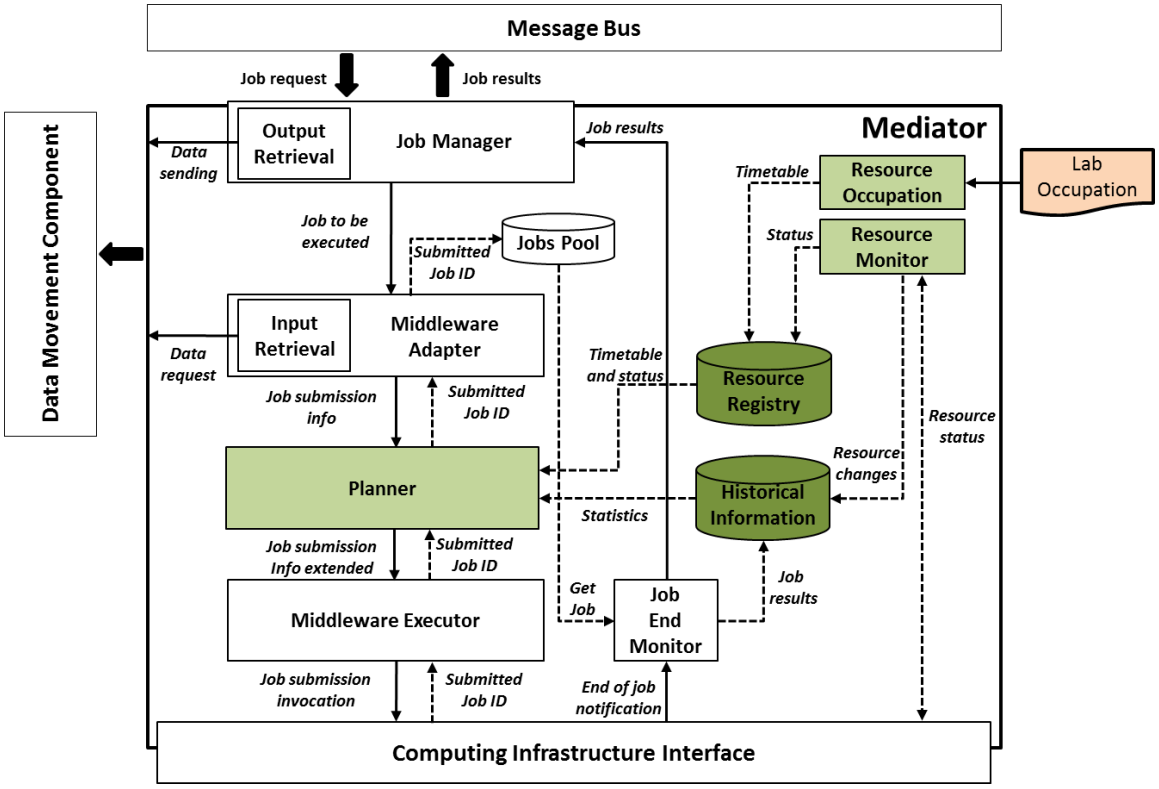


Figura 12: Esquema del mediador con funcionalidades añadidas

- ***Resource Occupation (Ocupación de Recursos)*:** Componente que recibe un fichero de texto con información de la ocupación de los recursos, lo traduce y lo almacena en el Resource Registry. Puede gestionar modificaciones en el horario. En este proyecto se ha configurado únicamente para recibir información de los laboratorios del Departamento de Informática e Ingeniería de Sistemas. Un fichero de ejemplo es el siguiente:

```
        L (24/03/14)      M (25/03/14)   F (26/03/14)  J (27/03/14)     V (28/03/14)
08..09  ________________  ___45a_______    Festivo     __345a5b__6b_  ___45a_______
09..10  1_3_____________  1_345a_______    Festivo     __345a5b__6b_  1__45a_______
10..11  1_34________R     1_345a______R    Festivo     12345a5b__6bR  1__45a______R
11..12  1_34________R     1234________R    Festivo     12345a______R  1__45a5b____R
12..13  12__5a______R     123_5a_______    Festivo     123_5a5b_____  1_345a5b_____
13..14  12__5a______R     1_3_5a_______    Festivo     123_5a5b_____  1_345a5b_____
14..15  ____5a_______     ____5a_______    Festivo     ______5b_____  ______________
15..16  1__45a______R     12__5a_______    Festivo     ____5a_______  ______________R
16..17  1__45a______R     123_5a__6a___    Festivo     __3_5a_______  ___4________R
17..18  123_5a______R     _2345a__6a__R    Festivo     1_3_5a_______  ___45a______R
18..19  123_________R     1234__5b6a6bR    Festivo     1_345a5b_____  1__45a______R
19..20  1_3_____________  1_34__5b6a6bR    Festivo     123_5a5b_____  1___5a_______
20..21  __3_____________  ___4__5b6a6bR    Festivo     123_5a5b_____  ______________
```

Este fragmento representa una semana de clase. Cada columna es un día de la semana; cada fila, una hora; y cada celda indica qué laboratorios están ocupados durante esa hora. En el apartado VI.3. Implementación de un mediador de recursos efímeros proporcionamos más detalles de implementación.

- ***Resource Monitor (Monitor de Recursos)*:** Componente que monitoriza de forma periódica el estado de los recursos (libre, ocupado, eliminado, etc.). Para ello, ejecuta el comando `condor_status –xml` en el nodo principal de HTCondor de forma periódica y configurable, extrae la información relevante y la almacena en el Resource Registry. Puede generar estadísticas de uso de los recursos.

- ***Resource Registry (Registro de Recursos)*:** Almacén que contiene el estado de los recursos en una tabla hash (procedente del Resource Monitor) e información acerca de su ocupación tanto pasada como futura en un mapa hash (proporcionada por el Resource Ocuppation). En ambos casos la información se almacena jerarquizada para facilitar su utilización, y de manera persistente en disco, gracias a la biblioteca MapDB de Java.

- ***Historical Information (Información Histórica)*:** Almacén de información que contiene información histórica sobre los trabajos ejecutados (procedente del Job End Monitor) y eventos referentes a la adición/eliminación de los recursos integrados en el entorno de computación (procedente del Resource Monitor), todo ello en tablas hash persistentes en disco. También permite obtener estadísticas sobre la utilización de los recursos y el rendimiento de los trabajos que puedan ser utilizadas durante la planificación. Este componente tiene un gran margen de desarrollo para trabajo futuro.

- ***Planner (Planificador)*:** Componente que permite seleccionar el recurso o grupo de recursos concretos en los que se ejecuta un trabajo, mediante la función `rank` de HTCondor. Recibe la información del trabajo a someter y añade a dicha información el recurso para ejecutar, obtenido mediante un algoritmo de planificación o *scheduling* que determina el lugar idóneo para su ejecución. Para ello, puede utilizar diferentes fuentes de información como son: el estado actual de los recursos y la ocupación prevista para los mismos (obtenidos del Resource Registry) y/o estadísticas de uso, tanto de los propios recursos como de ejecuciones pasadas de la misma aplicación (proporcionadas por el Historical Information).

En el Anexo VI – Integración con la Infraestructura del GIDHE describimos con mayor detalle tanto el diseño como la implementación, además de la gestión del almacenamiento de información.

Capítulo 5 – Verificación y análisis de la solución

Una vez instalado y configurado el sistema para nuestras necesidades, procedemos a realizar las pruebas pertinentes para comprobar su correcto funcionamiento. En este capítulo veremos un resumen de las pruebas realizadas. También realizaremos varias pruebas o experimentos para comprobar la efectividad del mediador de la Infraestructura, descrito en el capítulo anterior. En el Anexo IV – Pruebas de uso se encuentran detalladas todas las pruebas realizadas.

5.1. Pruebas del sistema de gestión de recursos efímeros

Para comprobar todos los posibles casos de fallo, amén de otros escenarios que podemos encontrarnos en la realidad, hemos preparado una serie de pruebas de uso con todos los datos necesarios para su realización.

En cada prueba, primero hemos planteado el proceso sobre el papel, los requisitos y el objetivo que perseguimos con su realización. Después, hemos preparado el entorno necesario, seguido de la ejecución de la prueba, y por último la evaluación de los resultados, comprobando que obtenemos lo que deseábamos.

Vamos a realizar tres tipos de pruebas: de funcionamiento general, de fallo y pruebas adicionales. En los siguientes apartados veremos un ejemplo de cada una de ellas. También veremos un resumen con los escenarios posibles de fallo que nos podemos encontrar.

5.1.1. Pruebas de funcionamiento general

El objetivo de estas pruebas es simular varias características en un entorno de funcionamiento normal, para comprobar que el sistema hace lo que se esperaba y cumple los requisitos establecidos. La tabla 2 muestra un ejemplo de una prueba de funcionamiento general en la que configuramos un sistema similar al del entorno real.

Título	**Prueba en condiciones reales**
Requisitos	6 instancias de Amazon EC2 [W5], una por cada laboratorio del Ada Byron, más el *Central Manager*. Todas ellas con el mismo rol y la misma configuración que queremos implementar en las máquinas reales.
Objetivo	Comprobar el correcto funcionamiento de todo el sistema en su conjunto, con la configuración definitiva.
Desarrollo	Iniciamos primero la instancia del *Central Manager*, y después todas las demás. Comprobamos con *condor_status* que se reconocen todas las máquinas. Sometemos varios trabajos desde el *Central Manager* (en local) y desde una de las otras máquinas (en remoto), para comprobar acto seguido la cola con *condor_q*.
Resultado	Correcto. Los trabajos se distribuyen entre todas las máquinas disponibles (cumpliendo los requisitos). Al acabar la ejecución, los trabajos sometidos en local se borran de la cola y se mandan los resultados al directorio desde el que se sometieron. Los sometidos remotamente se quedan, a la espera de ser recogidos.

Tabla 2: Prueba en condiciones reales

La prueba en condiciones reales es la que nos permitirá determinar que el sistema en su conjunto funciona realmente como habíamos planteado inicialmente.

5.1.2. Pruebas de fallo

En primer lugar, la tabla 3 muestra un resumen de los escenarios de fallo que pueden aparecer en el sistema. Para cada posible fallo se indican dos casillas, la primera indica el problema y la segunda, una posible solución. En el apartado IV.1. Escenarios posibles se describe los posibles casos de forma más textual y detallada.

Existen tres tipos caídas que pueden afectar a HTCondor.

1. Cambio de libre a ocupado. Un ordenador que estaba libre, deja de estarlo al llegar alguien que lo va a utilizar.
2. Apagado ordenado. Apagado correcto de la máquina, notificando al sistema.
3. Apagado brusco. Se da cuando el sistema se cuelga, cuando alguien apaga el ordenador desde el botón de encendido, quitando el enchufe, cuando se va la luz, etc.

	Cambio de libre a ocupado	Apagado ordenado	Apagado brusco
Central Manager	No le afecta.	Se notifica al admin y se deja de hacer *matchmaking*.	Se deja de hacer *matchmaking*.
	–	Configurar espejos para poder restaurarlo en otra máquina. *Alta disponibilidad*.	Configurar espejos para poder restaurarlo en otra máquina. *Alta disponibilidad*.
CCB Shared Port	No le afecta.	La máquina queda incomunicada, como si no estuviera.	La máquina queda incomunicada, como si no estuviera.
	–	Restaurar el demonio *condor_shared_port*	Restaurar el demonio *condor_shared_port*
Submit	No le afecta.	Los trabajos en ejecución terminan y esperan, pudiendo llegar a perderse.	Los trabajos en ejecución terminan y esperan, pudiendo llegar a perderse. Puede corromperse la cola.
	–	Levantar de nuevo el demonio *condor_schedd*	Levantar de nuevo el demonio *condor_schedd*
Execute	Se matan los trabajos que estuvieran ejecutando.	Migración, comenzando de 0 o de *checkpoint* reciente.	Migración desde 0
	Cambiar la política de ejecución, o mejorar el algoritmo de *scheduling*.	Pocas soluciones posibles. Encender la máquina o la plataforma HTCondor.	Pocas soluciones posibles. Encender la máquina o la plataforma HTCondor.

Tabla 3: Escenarios de caídas

A partir de estos datos, plantearemos para comprobar los casos más significativos. Buscaremos provocar situaciones de fallo, que se pueden dar en la realidad, y con los que nos tendremos que

enfrentar. El objetivo es ver cómo reacciona el sistema a cada uno de ellos. La tabla 4 muestra un ejemplo de prueba de fallo realizada, la que indica qué ocurre cuando cae el *Central manager*.

Título	**Caída del *Central Manager***
Requisitos	*Central Manager* funcionando, al menos una máquina *Execute* que ejecute un trabajo sometido por otra máquina *Submit*, y uno o más trabajos en cola.
Objetivo	Comprobar que los trabajos en ejecución terminan, y los que están en cola, permanecen en ella hasta que vuelve a estar disponible el *Central Manager*.
Desarrollo	Se dispone de un *Central Manager*, una máquina *Submit* y otra *Execute*, todas ellas independientes entre sí. Se someten varios trabajos desde el *Submit*, de los cuales al menos uno se empieza a ejecutar. En ese momento, apagamos el *Central Manager*, bien con `condor_off`, bien con `poweroff`. Vemos la cola de trabajos desde la máquina *Submit*, mediante `condor_q`.
Resultado	Satisfactorio. Los trabajos que esperan en la cola se quedan esperando, mientras que los que acaban, devuelven sus resultados. Pueden someterse nuevos trabajos, aunque no sean enlazados.

Tabla 4: Prueba caída del *Central Manager*

Gracias a este tipo de pruebas podemos hacernos a la idea de la magnitud del problema que supone cada fallo en el sistema.

5.1.3. Pruebas adicionales

Otras pruebas que debemos realizar, que al tratarse de casos especiales, no podemos englobar en casos de funcionamiento normal o de fallo. Con estos casos pretendemos simular la posibilidad de añadir nuevos recursos al sistema en un futuro. La tabla 5 muestra un ejemplo de prueba donde añadimos nuevas máquinas con IP privada.

Título	**Añadir máquinas con IP privada**
Requisitos	*Central Manager* conocido con IP pública, y servidor *CCB* activo. Permisos de lectura y escritura para las nuevas máquinas.
Objetivo	Comprobar que el *CCB* registra correctamente las nuevas máquinas privadas, y se establece una comunicación bidireccional entre ellas y el *Central Manager*.
Desarrollo	Esta prueba se ha realizado dos veces. Primero con una máquina virtual de CentOS de Amazon, no perteneciente a la misma red VPC que el *Central Manager*. Después, con un ordenador personal de casa, con IP privada y sistema operativo Windows 7. En ambas pruebas se ha configurado HTCondor de forma similar (cambiando los parámetros de red), introduciendo esta vez, las variables `CCB_ADDRESS` y `PRIVATE_NETWORK_NAME`.
Resultado	Satisfactorio. El *CCB* registra las máquinas y estas reconocen el *pool* y son accesibles para recibir trabajos.

Tabla 5: Prueba añadir máquinas con IP privada

Estas pruebas adicionales nos permitirán comprobar que el sistema es escalable, que es posible añadirle más recursos en cualquier momento para mejorar su productividad.

5.1.4. Conclusiones

Las pruebas realizadas nos han permitido comprobar que el sistema que simula los laboratorios funciona correctamente. Esto quiere decir que el despliegue en el entorno físico resultará sencillo, y no tendrá grandes complicaciones.

Con las pruebas realizadas sobre características adicionales y sistemas especiales, podemos concluir dos cosas. Primero, que el sistema es fácilmente escalable, puesto que admite nuevos recursos y puede interactuar con ellos sin problema. Y también, que se trata de una de una solución general, es decir, portable a otros entornos distintos.

5.2. Pruebas del mediador de recursos efímeros

Debido a la gran cantidad de componentes existentes en el mediador, y a la complejidad de los mismos, ha sido necesario hacer pruebas individuales. Las pruebas principales que hemos realizado han sido las siguientes:

- Comprobar que el mediador recibe, traduce y almacena el horario de ocupación de los laboratorios correctamente, y de manera persistente.
- Comprobar que el mediador se comunica correctamente con el colector de HTCondor, ejecutando el comando `condor_status –xml`, recibe la información sobre el estado de los recursos actuales, la traduce y la almacena de manera persistente.
- Comprobar que el mediador reconoce los recursos que están y van a estar libres a corto plazo, a partir del horario de los mismos previamente suministrado.

Una vez que comprobamos que todos los componentes funcionan correctamente por separado, hemos realizado una prueba similar al experimento que se lanzaría más tarde, a través de la herramienta Renew [W28]. Este experimento consiste en una red de Petri de Alto Nivel que espera durante una hora y luego lanza un número conocido de tareas (mediante el comando `condor_submit`). En esta prueba sin embargo eliminaremos el tiempo de espera.

El objetivo de dicha prueba no es otro que el de comprobar que todo lo que se ha desarrollado previamente funciona correctamente de manera conjunta, evitando las largas esperas que tienen los experimentos definitivos.

5.3. Evaluación del mediador de recursos efímeros

En este apartado vamos a evaluar la utilidad del mediador de recursos efímeros desarrollado mediante una prueba de concepto de ejecución de diferentes tareas computacionales sobre un entorno que simula una situación real de ocupación de los laboratorios del DIIS. Para ello, vamos a describir el entorno de simulación creado y los experimentos que se van a ejecutar para evaluar el mediador desarrollado. Finalmente, evaluaremos los resultados y presentaremos las conclusiones que se extraen de los mismos.

5.3.1. Descripción del entorno

Para la realización de la evaluación, hemos tomado como entorno de referencia el entorno creado anteriormente en Amazon EC2, como hemos visto en el apartado 4.2. Esquema general de despliegue en Amazon, con una máquina virtual por cada laboratorio real. Partiendo de dicho entorno, hemos cambiado el nodo central virtual por el ordenador personal del I3A para mostrar que no es necesario que el mismo se encuentre en el mismo entorno. Para ello hemos modificado en el fichero de configuración de las máquinas ejecutables, la dirección del *Central Manager*. Y para que éste reconociese las nuevas máquinas, al ser máquinas privadas a los ojos del *Central Manager*, hemos añadido en la configuración de las máquinas, un parámetro para que se registrasen en el *CCB*.

Para simular la ocupación de los laboratorios, hemos creado un horario ficticio comenzando con la hora actual y las siguientes, con el formato reconocido por el componente HTCondorResourceOccupation del mediador, y lo hemos introducido en la base de datos, para que pudiera ser consultado por el componente de planificación. Hemos configurado el horario de ocupación de los laboratorios según se muestra en la tabla 6. Las horas de ocupación de cada laboratorio se muestran en rojo; las horas libres, en verde.

	lab001	lab002	lab003	lab004	lab005	lab102
1ª hora						
2ª hora						
3ª hora						
4ª hora						

Tabla 6: Horario de ocupación de los laboratorios

Para simular las caídas de los laboratorios, hemos programado como tareas la ejecución del comando `service condor stop` cada vez que un laboratorio pasaba a estar ocupado, y el comando `service condor start` cuando éste pasaba a estar libre. Esto ha sido posible gracias a la herramienta `at` de Linux. En todos los experimentos realizados se ha supuesto que la ocupación de los laboratorios era exactamente la misma, para poder compararlos de forma justa.

La principal función del planificador es la de seleccionar las máquinas libres antes que las ocupadas a la hora de ejecutar tareas. Para ello, hemos utilizado la variable RANK de HTCondor.

5.3.2. Descripción del experimento

Para la evaluación de la efectividad del mediador, hemos realizado 4 experimentos: 2 de ellos en los que se ejecutan “tareas largas”, primero con planificador y luego sin él; y otros 2 en los que se ejecutan “tareas cortas”, también con y sin planificador. Respecto de las tareas ejecutadas, su única misión es la de simular carga computacional durante el tiempo estipulado, de forma que, el contenido concreto de las tareas que se ejecutan no resulta interesante para la realización del experimento. Gracias a esto sabemos exactamente cuánto les va a costar ejecutarse y podemos realizar los experimentos bajo las mismas condiciones.

En el experimento con “tareas largas”, hemos lanzado 2 tareas cada hora durante 4 horas, de 50 minutos de duración cada una. En el experimento con “tareas cortas”, hemos lanzado 10 tareas

cada hora, también durante 4 horas, de 10 minutos de duración. Por tanto, en ambos casos el tiempo de ejecución es el mismo y es comparable.

El objetivo de realizar dos tipos de experimento (con “tareas largas” y con “tareas cortas”), es evaluar cuál es el tipo de tarea que mejor se adapta a las condiciones del entorno de ejecución y cuál es la efectividad del mediador desarrollado ante diferentes tipos de tareas.

Un dato que conviene destacar es que cada remesa de tareas se ha sometido 1 minuto antes de la hora siguiente. Y también, que los laboratorios, cuando el horario decía que pasaban a estar ocupados, se caían pasados 5 minutos de la hora. De esta forma, facilitábamos las expulsiones de trabajos cuando llevaban poco tiempo ejecutado, en el caso de no haber sido bien repartidos. Después, cada máquina vuelve a estar disponible 5 minutos antes de terminar la última hora de ocupación, según el horario.

Finalmente, cabe añadir que la ejecución de los experimentos se ha automatizado utilizando la infraestructura de computación del GIDHE mediante el diseño de una red de Petri de Alto Nivel [B1] en la herramienta Renew.

5.3.3. Resultados

Una vez realizados los experimentos se han analizado los ficheros de *log* devueltos al término de las ejecuciones y hemos comparado los resultados de los mismos. La tabla 7 muestra un resumen de los resultados obtenidos. En ella podemos ver el número total de tareas ejecutadas, (8 largas primero y 40 cortas después), la duración de cada una, la cantidad de tareas que se sometían simultáneamente cada hora, y los resultados obtenidos a partir de los *logs*.

	Tareas largas		Tareas cortas	
	Con planificador	Sin planificador	Con planificador	Sin planificador
Nº de tareas	8		40	
Duración	50 min		10 min	
Nº de tareas simultáneas	2		10	
Nº de expulsiones	0	3	5	5
Tiempo perdido	–	33 min	55 min	55 min

Tabla 7: Evaluación de resultados

El **número de expulsiones** consiste en el número de tareas, del total que se han sometido, que no han podido terminar su ejecución la primera vez y han tenido que ser reubicadas. El **tiempo perdido** de una tarea es el tiempo de ejecución que ha transcurrido antes de ser expulsada, junto con el tiempo de reubicación. Cada casilla corresponde a la suma del tiempo perdido entre todas las tareas expulsadas (3 con tareas largas sin planificador, y 5 en cada caso de tareas cortas).

Por la forma en que se ha programado el experimento, el tiempo perdido de ejecución son 6 minutos, desde que se ejecuta hasta que es expulsada. El tiempo de reubicación es variable,

depende de HTCondor. Incluye el tiempo que tarde en volver a encolar ese trabajo en el *Central Manager*, encontrar un nuevo recurso libre y enviarlo. En media, el tiempo que hemos observado es de 5 minutos.

A continuación analizaremos caso por caso.

- **Prueba de tareas largas con planificador.** En ese caso el planificador, antes de mandar cada remesa de tareas, ha añadido las máquinas libres como prioritarias. Es decir, si hay posibilidad de decisión entre una máquina prioritaria y otra normal, se elegirá la primera. Y como el número de tareas no excede en ningún momento al de recursos disponibles, todas las tareas se han ejecutado en máquinas libres (prioritarias), por lo que no ha habido expulsiones.

- **Prueba de tareas largas sin planificador.** Dado que en este caso no se establecen prioridades, HTCondor selecciona aleatoriamente la máquina para ejecutar. En este caso, de entre todas las tareas que se han sometido en total, 3 de ellas han ido a parar a una máquina que 6 minutos más tarde se iba a caer. La primera de las expulsiones se ha dado en la tarea que ha intentado ejecutar en lab001en la primera hora; la segunda, en lab004 durante la segunda hora; la última, en lab002 durante la tercera hora.

- **Prueba de tareas cortas con planificador.** En este caso, el planificador añade como recomendaciones los laboratorios que hay libres en ese momento. Sin embargo, al someterse un número de tareas mayor que el número de recursos disponibles, primero se mandarán a las máquinas libres (prioritarias), y cuando no queden más, se probará con las demás, pero en todos los casos se acabará expulsando a la tarea. Esto ocurrirá cada vez que provoquemos la caída de una máquina: durante la primera hora, en lab001 y lab003; durante la segunda, en lab004 y lab102; y durante la tercera, en lab002. En total 5 caídas, y como hay más trabajos que recursos, 5 expulsiones.

- **Prueba de tareas cortas sin planificador.** Aquí ocurre lo mismo que en el caso anterior, con la salvedad que en este caso no se hace recomendación inicial. En todo momento se eligen las máquinas aleatoriamente. Y como sigue habiendo más tareas que recursos, cada vez que haya una caída en una máquina, habrá una expulsión.

5.3.4. Conclusiones

Tras la ejecución de las pruebas y su evaluación, llegamos a la conclusión de que el planificador implementado en el mediador de recursos efímeros es de utilidad en este tipo de entornos y puede ahorrar tiempo de ejecución, principalmente cuando el número de tareas a ejecutar es menor al número de recursos libres.

El motivo es que el mediador conoce el horario de la ocupación de los recursos, y evita en la medida de lo posible, mandar trabajos a ordenadores que van a ser ocupados, y que por tanto los trabajos sean expulsados. Esto se cumple en el experimento de las tareas largas, pues hay un número menor o igual de que de recursos disponibles.

Sin embargo, en el caso de tareas cortas, al ejecutarse un número de tareas mayor que el número de recursos, y añadiendo el hecho de que cada máquina puede ejecutar una sola tarea cada vez,

resulta imposible que la recomendación del mediador respecto a los recursos libres se cumpla. Da lo mismo mandar un trabajo a un laboratorio que lo va a expulsar antes de que termine, que dejarlo en la cola esperando. Hay que tener en cuenta que en un caso real, el hecho de que un laboratorio esté ocupado no implica que todos los ordenadores vayan a ser utilizados y por esa razón el envío de tareas a recursos en laboratorios ocupados podría ahorrar tiempo.

Finalmente, dichos experimentos muestran que el número de recursos libres y su ocupación influye enormemente en si es preferible ejecutar tareas cortas o tareas largas. En general, es deseable ejecutar tareas cortas porque aprovechamos todos los recursos disponibles y en caso de que un recurso en el que estamos ejecutando falle, la pérdida en tiempo es potencialmente menor que en el caso de ejecutar tareas largas. Sin embargo, la ejecución de tareas más largas en recursos que sabemos que van a estar disponibles puede beneficiarnos al evitar la aparición de fallos. En conclusión, lo óptimo sería utilizar un número de tareas igual al número de recursos libres en cada momento, para aprovechar al máximo los recursos.

Capítulo 6 – Conclusiones y trabajo futuro

En este capítulo presentaremos las conclusiones obtenidas de la realización del proyecto. Analizaremos el cumplimiento de los objetivos, los problemas encontrados, y plantearemos varias propuestas de trabajo futuras. Para terminar, se muestra una valoración personal del trabajo realizado.

6.1. Conclusiones

En el presente proyecto se ha llevado a cabo la configuración y el despliegue de la plataforma HTCondor [B10] (previamente seleccionada tras un estudio previo), para la gestión de recursos efímeros. La solución propuesta se ha verificado con éxito en un entorno virtual en Amazon EC2 [W6], abriendo la posibilidad de que se pueda desplegaren un entorno real en el ámbito académico de forma inmediata. También se ha integrado dicha plataforma con la Infraestructura del grupo de investigación GIDHE [W2], mediante la extensión de un mediador para la gestión de recursos efímeros, añadiendo potencia computacional a la Infraestructura.

Por lo general se han cumplido todos los objetivos propuestos, salvo el relativo al despliegue en el entorno real. En cuanto concluimos que la plataforma a utilizar sería HTCondor, notificamos a los administradores nuestra intención de instalarla en los laboratorios de prácticas ante lo que nos mostraron su predisposición. Una vez que nuestro sistema había sido ampliamente probado y funcionaba correctamente, acordamos con los administradores que ellos instalarían HTCondor en el sistema para, acto seguido, darnos permisos para modificar los ficheros de configuración (el instalar nosotros la plataforma no era ni siquiera una posibilidad). Sin embargo finalmente no fue posible este acceso, lo que nos obligó a replantear el entorno de prueba final, con el nodo central en un ordenador personal del I3A, y simulando los laboratorios como máquinas virtuales que se conectaban a éste.

En cualquier caso, lo que se ha conseguido demuestra que el sistema funciona y que podría funcionar como se propuso inicialmente si se dieran las circunstancias necesarias.

Las aportaciones de este proyecto han consistido en el estudio y comparativa de los diferentes tipos de computación (*cluster*, *grid* y *cloud*) junto con las plataformas más importantes; en la configuración de un sistema escalable capaz de aprovechar recursos efímeros y proporcionar una potencia computacional que de otra manera estaría desaprovechada; en la inclusión en dicho sistema de algunas características novedosas como la autenticación por SSL, de las que apenas existe información útil, lo que supone el aporte de una información que actualmente no existe o no está bien definida.

La principal novedad que ha tenido este proyecto respecto a otros ha sido la simulación de un entorno real en una plataforma *cloud* virtual como es Amazon EC2. Este hecho tiene numerosas ventajas. Entre las más importantes está el hecho de tener total libertad de actuación sobre el entorno virtual para probar las características a implementar, sin riesgo de ningún tipo sobre el entorno real y el ahorro de tiempo a la hora del despliegue en el entorno real, al hacerlo ya con una solución probada y funcional. También cabe destacar la mejora de planificación de trabajos obtenida con la integración de la plataforma HTCondor a un mediador de la Infraestructura del GIDHE.

6.2. Trabajo futuro

La principal línea de trabajo futuro es el despliegue en un entorno real. Es uno de los principales objetivos de este proyecto: el planteamiento de un sistema de gestión de recursos efímeros, para su posterior despliegue en un entorno real.

Se podría configurar la función *GlideIn* de HTCondor. Esto nos daría la posibilidad de ejecutar trabajos en otros recursos con otro *middleware* compatible con la herramienta Globus. También, para la ejecución de trabajos intensivos en datos (los tradicionales son intensivos en CPU), HTCondor dispone de la extensión Stork. También se podría configurar en nuestro sistema.

Sobre la Infraestructura, el principal margen de desarrollo está en el Planificador (*Planner*), y más concretamente, en el algoritmo de planificación o *scheduling*. Es imposible resolver el problema de planificación en un tiempo razonable, por lo que tenemos que hacerlo por aproximaciones, algunas de ellas muy complejas. De hecho, todas las mejoras que se implementen, irán con el objetivo de mejorar este algoritmo. Otro elemento con un gran margen para desarrollo es el encargado de recopilar las estadísticas para futuros usos.

Además de las ya mencionadas, algunas líneas de trabajo futuro para la Infraestructura son las siguientes:

- Añadir compatibilidad para monitorizar otras organizaciones que añadamos al sistema.
- Recopilar estadísticas de los *logs* devueltos por trabajos finalizados.
- Obtener estadísticas de utilización de laboratorios: dentro y fuera de prácticas.
- Añadir la posibilidad de establecer un tiempo aproximado de ejecución, para un reparto más eficiente de las tareas.
- Añadir la posibilidad de que el mediador agrupe varias tareas cortas en tareas de una mayor duración para mejorar el aprovechamiento de los recursos y disminuir las expulsiones.

6.3. Valoración personal

Cuando llegó el momento de elegir PFC, no tenía nada claro sobre qué quería realizar. Después de haber acabado casi todas las asignaturas, no había descubierto ningún ámbito de la informática que me gustara por encima de todos los demás.

Ha sido en la última parte de mi carrera en la que he ido teniendo cada vez más claro por donde quería que fuese mi futuro profesional. Me gustaba sobre todo el tema de la seguridad informática y de administración de sistemas, incluyendo el tema de redes. Por eso, tras semanas de búsqueda, finalmente me decanté por este proyecto. Entre las razones que me llevaron a tomar esta decisión fue la posibilidad de probar algo relativamente nuevo como lo era el hecho de crear un entorno virtual en Amazon EC2.

A pesar de que el proyecto, entre unos motivos y otros, se ha alargado en el tiempo más de lo que hubiese deseado, estoy satisfecho con todo lo que he aprendido. Ha habido momentos de pensar que no tenía fin, porque cada vez que se terminaban unos requisitos iban surgiendo

nuevas ideas y nuevas vías para continuar, y llegaba un momento en el que había que cortar por algún lado.

La mayor complicación que he tenido durante el proyecto ha sido la poca información útil que he encontrado sobre HTCondor. A pesar de tener un manual de casi 1000 páginas, hay temas como por ejemplo la autenticación por SSL que al ser menos utilizados por la comunidad, apenas se explican.

Finalmente, una de las cosas más importantes que he aprendido con la realización de este proyecto es la importancia de una buena metodología y planificación, gracias a la cual he podido terminar de forma satisfactoria este proyecto, pudiendo incluso sortear de forma efectiva los problemas que han ido apareciendo.

Anexo I – Estado del arte extendido

Recientemente, debido a la gran demanda de recursos computacionales para el procesamiento de trabajos que requieren cientos o miles de procesadores, para aprovechar la potencia de grandes grupos de computadores se han desarrollado diferentes tipos y paradigmas de computación.

A diferencia de la computación monolítica (la forma tradicional, en un solo computador físico), la computación distribuida es un término bastante ambiguo, con muchos enfoques y paradigmas distintos que lo componen.

Actualmente existen tres tipos distintos de computación distribuida: computación *cluster*, *cloud*, y *grid* [B4]. Cada uno resuelve una necesidad. Dependiendo de los recursos de los que disponemos y del problema que queremos resolver, para seleccionar el paradigma adecuado, necesitaremos hacer un estudio exhaustivo de los tipos de plataformas que podemos encontrarnos hoy en día, y ver cuál es la que resuelve mejor nuestro problema.

En este anexo vamos a ver los tipos o enfoques de la computación distribuida, los paradigmas más importantes y algunas plataformas, con el objetivo de compararlos, analizando sus ventajas e inconvenientes, para finalmente quedarnos con la plataforma que mejor nos convenga para éste proyecto.

I.1. Computación *cluster*

La computación *cluster* [B5] es el tipo más sencillo de computación distribuida, y también el más antiguo. Cuando hablamos de un clúster de ordenadores, nos referimos a un grupo de ellos trabajando conjuntamente para un mismo fin. Se trata pues de ordenadores con dedicación exclusiva para resolver los problemas que se le encarguen desde un ordenador administrador.

Un clúster bien puede ser un supercomputador formado por un montón de nodos organizados en racks, o bien, varios ordenadores personales normales y corrientes. En ambos casos, están unidos todos los nodos por switch en una misma red local y tienen un nodo superior, que es el que controla todos los demás nodos. También disponen de un software que realiza la distribución de la carga de trabajo entre los equipos.

Es una alternativa a un equipo multiprocesador. La principal ventaja de este modelo de computación es la escalabilidad y el aumento de la relación coste/rendimiento. Para añadir más potencia/almacenamiento/etc., nos basta con añadir más equipos.

Como desventajas está el hecho de que los computadores que forman parte del clúster tienen dedicación exclusiva, no se pueden utilizar para otro fin que el de resolver tareas. Y además, la programación y el mantenimiento de estos sistemas son costosos.

Ejemplos: MOSIX [W8], PBS [W7], LSF [W9] o Grid Engine [W10] (los dos últimos en versión clúster).

I.1.1. Arquitectura *cluster*

En la figura 13 podemos ver cómo sería un clúster tradicional de ordenadores.

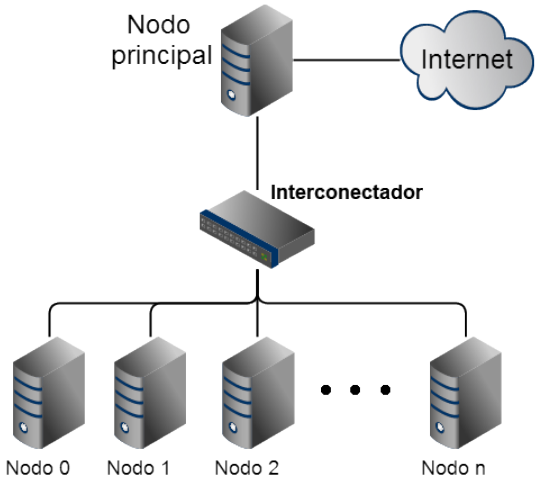


Figura 13. Arquitectura *cluster*

El nodo principal es el encargado de suministrar tareas a los nodos de computación, dispuestos en una red local. El nodo principal es accesible a través de internet, mientras que los demás, sólo son accesibles a través del principal.

Para conseguir más potencia de computación, basta con añadir más nodos de computación a la red local, ya sean racks profesionales u ordenadores personales.

Se trata de un sistema SSI (Single System Image). En un sistema SSI todas las computadoras vinculadas dependen de un sistema operativo común, diseñado al efecto.

I.1.2. *Middlewares cluster*

Un *middleware* de *cluster* es un software que actúa como intermediario para dar una visión de homogeneidad al usuario, en un sistema heterogéneo. Aquí vamos a analizar dos de las plataformas *cluster* más importantes: openMosix y PBS.

I.1.2.1. openMosix

openMosix [W6] tiene su origen en MOSIX, un sistema de clustering SSI (Single System Image), cuyo desarrollo comenzó en 1981 en la Hebrew University of Jerusalem. Aunque al principio se basaba en el sistema UNIX, en 1999 fue portado a Linux. En 2001 MOSIX se convierte en software propietario, y es entonces cuando nace openMosix, la versión abierta de MOSIX.

openMosix es un software que permite que ordenadores conectados en red funcionando bajo GNU/Linux trabajen de forma cooperativa. Es capaz de balancear automáticamente la carga entre los diferentes nodos del clúster y permite la adición o sustracción de nodos en caliente sin

necesidad de interrumpir el servicio. La carga se distribuye entre los distintos nodos atendiendo a parámetros como tipo de conexión, memoria disponible y velocidad de CPU.

Dado que openMosix forma parte del kernel y mantiene total compatibilidad con Linux, los programas de usuario, ficheros y otros recursos funcionarán igualmente sin cambio alguno. Para el usuario final, todo el clúster se presenta como un gran sistema multiprocesador. Todos los nodos de un clúster openMosix cuentan con el mismo kernel modificado, de ahí el nombre de clúster Single System Image (SSI).

El algoritmo de balanceo de carga migra los procesos entre los distintos nodos de forma transparente, intentando optimizar su utilización en todo momento, si bien el administrador puede modificar manualmente este algoritmo.

El hecho de que la migración de procesos se haga de forma transparente hace que el conjunto del clúster se muestre como un gran sistema multiprocesador (SMP) con tantos procesadores disponibles como el sumatorio de los procesadores de todos los nodos [B5].

I.1.2.2. PBS y OpenPBS

PBS (*Portable Batch System*) [W7] es un sistema de trabajos por lotes y un sistema de gestión de recursos. Fue desarrollado por la *NASA* a principios de los años 90 [B5] .

PBS consta de cuatro componentes:

- **Commands**: Permiten enviar, monitorizar, modificar y borrar trabajos.
- **Job Server**: La parte central de PBS. Proporciona los servicios básicos como recibir y crear un trabajo, modificarlo, protegerlo de posibles caídas del sistema y ponerlo en ejecución.
- **Job Executor**: Pone en ejecución el trabajo cuando recibe una copia del servidor. También reproduce la sesión del usuario propietario del trabajo (shell, .login, .csh, etc). Finalmente, al acabar el trabajo, devuelve la salida al usuario.
- **Job Scheduler**: Contiene las políticas que controlan qué trabajo, cuándo o dónde se ejecutan los trabajos.

Se trata de un software propietario de la empresa Altair. Existen varias versiones. Las principales son PBS Professional (de pago) y PBS Analytics.

I.2. Computación *grid*

La computación *grid* [B6] [B7] es un tipo de computación más abstracto. Su principal objetivo es el mismo que el de la *cluster*, es decir, aprovechar recursos para resolver problemas, de cómputo o de almacenamiento. Algunas claves que cambian en un entorno *grid* respecto a un clúster son la posibilidad de gestionarlo de forma descentralizada, el tipo de recursos que se gestionan, heterogéneos y normalmente no dedicados (aprovechando únicamente su tiempo libre), y el mayor alcance, de mayor dispersión geográfica que un clúster.

Es una tecnología que permite utilizar de forma coordinada todo tipo de recursos (computación, almacenamiento y aplicaciones) que no están sujetos a un control centralizado. A diferencia de la computación *cluster*, con la computación *grid*, los recursos pueden ser heterogéneos, y pueden estar distribuidos tanto en una pequeña red local, como en todo internet.

Como no se trata con recursos con dedicación exclusiva, como en el caso de *cluster*, sino que se utilizan sólo en su tiempo libre (se podría cambiar su configuración), el coste que tienen es mínimo. Esto favorece enormemente a la comunidad científica, y a cualquiera que desee hacer uso, máxime en tiempos de crisis, cuando los recursos son más limitados.

I.2.1. Arquitectura *grid*

La figura 14 muestra las diferentes capas del protocolo *grid*, y la interacción entre ellas y el nivel de aplicación [B12].

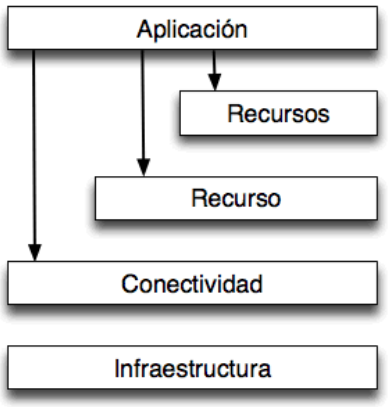


Figura 14: Arquitectura *grid*

- **Capa *Infraestructura*.** Aquí se encuentran los recursos físicos: ordenadores personales, clústeres, sistemas de almacenamiento, etc. También se incluye la infraestructura de red y sus mecanismos de gestión y control.

- **Capa *Conectividad*.** Incluye los protocolos de comunicación y de seguridad que permiten la comunicación entre los recursos computacionales. Entre ellos podemos encontrar la pila de protocolos TCP/IP, protocolos en redes de alta velocidad, SSL o Certificados X.509. Esta capa es especialmente importante ya que en la computación Grid intervienen recursos heterogéneos, posiblemente de distintas organizaciones con distintas políticas de seguridad.

- **Capa *Recurso* o de recursos individuales.** Incluye servicios y protocolos para el control y gestión de recursos individuales. Existen dos tipos de protocolos principales:

    - *Protocolos de información.* Permiten obtener información sobre un determinado recurso (características técnicas, carga actual, número de procesadores, memoria disponible, etc.).

    - *Protocolos de gestión.* Permiten el control de un determinado recurso: acceso, arranque, parada, monitorización, contabilidad o auditoria del recurso.

- **Capa *Recursos* o de recursos colectivos.** Aquí se encuentran los servicios que gestionan conjuntos de recursos. Los más comunes que se pueden encontrar en esta capa son:
    - *Servicios de directorio.* Permiten descubrir los recursos de organizaciones virtuales, así como consultar sus propiedades.
    - *Servicios de planificación y asignación.* Permiten la correcta asignación de cada tarea a un recurso.
    - *Servicios de monitorización y diagnóstico.* Informan sobre el estado de los recursos del Grid.
    - *Servicios de contabilidad.* Realizan operaciones de cálculo del coste de la utilización de varios recursos heterogéneos.
    - *Servicios de gestión de datos*. Gestionan las bases de datos necesarias en cada recurso.
- **Capa *Aplicaciones*.** Aquí están las aplicaciones que se ejecutan en la infraestructura *grid*. Según las exigencias de la aplicación, puede ser necesario pasar por todas ellas, o conectarse directamente a la infraestructura, depende de cada una. Los principales tipos son:
    - *Supercomputación distribuida.* Son aplicaciones cuyas necesidades son imposibles de satisfacer por una única organización. Requieren demandas puntuales e intensivas de computación. Ejemplos: simulaciones de complejos fenómenos físicos o cálculos numéricos.
    - *Sistemas distribuidos en tiempo real.* Son sistemas que generan un flujo de datos a alta velocidad que debe ser analizado en tiempo real. Ejemplos: e-Medicine, experimentos de física de alta energía (LHC) y control remoto de un recurso no-trivial (microscopios o equipo médico).
    - *Servicios puntuales*. Son semejantes a las dos aplicaciones anteriores, pero se diferencian en que estos servicios no se refieren a "potencia computacional" y no tienen por qué ser en tiempo real. Un ejemplo de este tipo de aplicación es el acceso a hardware específico para ciertos tipos de análisis (químico o biológico).
    - *Procesos intensivos de datos*. Son aplicaciones que trabajan con grandes volúmenes de datos y que son imposibles de almacenar en un único nodo. En su lugar, los datos se distribuyen a lo largo del Grid.
    - *Entornos virtuales de colaboración*. Este tipo de aplicaciones utilizan los recursos *grid* para crear entornos virtuales en 3D distribuidos.

I.2.2. Paradigmas *grid*

La definición de computación grid es muy general y abarca un elevado número de escenarios y situaciones. En la práctica, existen diferentes paradigmas de computación grid que cubren situaciones más específicas y satisfacen necesidades concretas.

A continuación vamos a ver los distintos paradigmas de computación grid que hay actualmente. Todos ellos tienen un enfoque común, el de aprovechar los recursos distribuidos. La diferencia vendrá sobre todo en el alcance de los recursos y en su organización.

I.2.2.1. Paradigma intranet computing

Al contrario que el paradigma de computación *cluster*, las herramientas de computación en *intranet* realizan una planificación basada en la suposición de que los recursos no están dedicados, y por tanto no tienen por qué estar disponibles durante toda la ejecución de los trabajos. Normalmente se usan para aprovechar los recursos en la intranet de la empresa o del centro de investigación sin salir de su dominio de administración.

El objetivo del paradigma *intranet computing* , también llamado *Institutional Desktop Grid*, es unir la potencia computacional desaprovechada de los recursos hardware distribuidos dentro de un único dominio de administración, para alcanzar rendimientos semejantes a los proporcionados por los sistemas de alto rendimiento comerciales con un coste diferencial prácticamente nulo.

Estas herramientas suelen instalarse junto con otros servicios que permiten compartir ficheros, como NFS, y usuarios, como NIS. Además, opcionalmente, se pueden instalar mecanismos de encriptación y autenticación que utilicen por ejemplo Kerberos o Diffie-Hellman. Intranet Computing es la alternativa más eficiente para aprovechar la capacidad de procesamiento disponible en los equipos ociosos de la red, por medio de la ejecución de trabajos independientes, normalmente programas paralelos o paramétricos.

Ejemplos: HTCondor [B10], Grid Engine y LSF (en versión *grid*)

I.2.2.2. Paradigma internet computing

Los modelos de *cluster* e *intranet computing* suelen ser efectivos a pequeña escala, pero si intentamos extrapolarlos a la red de redes, internet, entra en juego la escasa seguridad existente, lo cual es un problema [B6].

Este modelo de computación, también conocido como *Volunteer Computing*, lo que pretende es abarcar recursos no sólo dentro de un clúster o una red privada, sino en toda la red de redes, internet. Para ello utilizará un modelo similar al de cliente/servidor, obteniendo de esta manera un gran rendimiento en la ejecución de aplicaciones.

Aunque la productividad por cada ordenador será bastante más baja, se compensa al tener muchas más máquinas a su disposición. Esto es debido a que en este paradigma, se deben tolerar a los *hackers*, porque es imposible verificar cada una de las máquinas distribuidas por todo el mundo. Para ello hay que añadir redundancia (ejecutar la misma tarea en 2 o 3 procesadores, para contrastar y comprobar que el resultado es correcto), lo que le resta mucha potencia de cálculo. Hay que estudiar en cada caso si realmente compensa o no.

La idea principal de este paradigma es que cualquiera, donde quiera que esté, pueda donar tiempo de CPU y almacenamiento. Esto se consigue con un cliente que el usuario instala en su

ordenador, que pedirá y ejecutará las tareas que le manden. Algunas aplicaciones pueden ser investigaciones astronómicas o estudios de enfermedades como el cáncer.

Ejemplos: BOINC [W13], SZTAKI Desktop Grid [W14], etc.

I.2.2.3. Paradigma peer-to-peer computing

Las soluciones de computación indicadas anteriormente se basan en el modelo cliente/servidor. Existe un servidor o cliente principal que reparte trabajo entre los servidores o trabajadores, bien sea "tirando" (*pull*) o "empujando" (*push*). El principal problema que tienen es el cuello de botella que se suele formar en los servidores de sistemas masivos [B12].

Un sistema *peer-to-peer* (P2P) es un sistema *grid* especial. Puede reunir recursos heterogéneos distribuidos por todo el mundo, sin un servidor central. Dicha descentralización supone una solución al cuello de botella del modelo cliente/servidor [B14].

Este modelo es muy común en la compartición de ficheros en entornos distribuidos, por ejemplo, con el protocolo bittorrent. Sin embargo, el hecho de compartir ciclos de procesador está menos extendido.

Uno de los mayores desafíos con los sistemas P2P es permitir a los dispositivos encontrarse entre sí en un paradigma de computación que carece de un punto central de control, y con un número potencialmente enorme de pares (nodos), descentralizados y situados a nivel mundial.

Además, como los PC u otros recursos individuales se conectan a Internet voluntariamente, el grupo de recursos formado por estos PC no es constante. Por lo tanto, la tolerancia a fallos es también un desafío típico en los sistemas P2P.

Ejemplos: XtremWeb-CH [W15].

I.3. Plataformas *grid* más importantes

Ahora vamos a analizar las plataformas más importantes que hay actualmente. Vamos a analizar las más importantes, y más en profundidad las que están disponibles de forma gratuita.

I.3.1. BOINC

BOINC (**B**erkeley **O**pen **I**nfrastructure for **N**etwork **C**omputing) [W13] es el sistema ***middleware*** open source más utilizado y aceptado para **computación en internet**. La idea básica de este proyecto es la de crear una plataforma común que permita el desarrollo y ejecución de diferentes proyectos sin que sea necesario partir de cero en su desarrollo. Especialmente para usar los ciclos de CPU y GPU no utilizados. Se controla mediante RPC (*Remote Procedure Calls*, línea de comandos remota) o por el administrador de cuentas BOINC.

Arquitectura

La figura 15 muestra el esquema de la arquitectura del sistema BOINC.

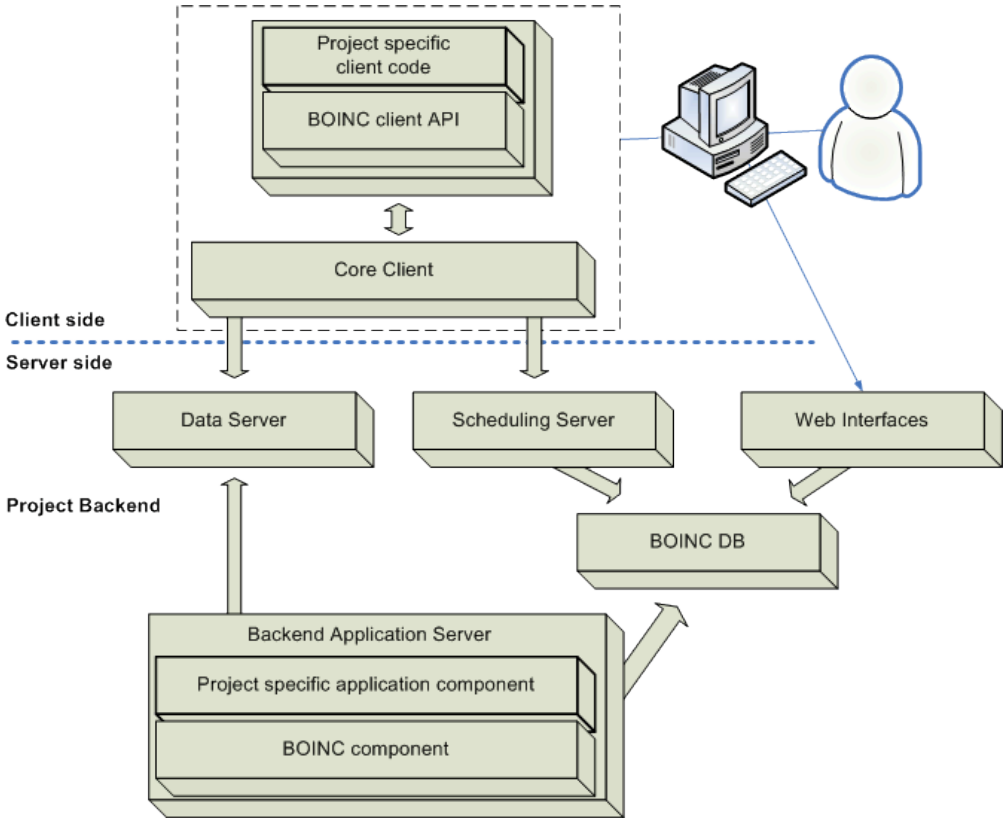


Figura 15: Arquitectura de BOINC [W29]

El **cliente** de BOINC es muy sencillo. Permite elegir al usuario los proyectos a los que desea apoyar, y los recursos que está dispuesto a ceder. Hecho esto, el programa cliente pregunta por tareas y se pone a ejecutarlas.

El **servidor** es más complejo. Nunca toma la iniciativa, tiene que ser el cliente el que le pregunte primero siempre. Gestiona las actualizaciones, los proyectos y las tareas.

Funcionamiento

1. El PC recibe un conjunto de tareas del servidor de tareas del proyecto. Estas tareas dependen de la PC, tiene en cuenta cuánta RAM tenemos, cuánto CPU disponible, etc.
2. El servidor le envía el programa a ejecutar y los datos de entrada. Si el servidor tiene una nueva versión de la aplicación, le va a mandar el ejecutable automáticamente.
3. Ejecuta la aplicación con los datos de entrada proporcionados, y genera un resultado.
4. Sube los ficheros de resultados al servidor de datos.
5. Comunica al servidor que completó la tarea que le asignaron, y recibe nuevos trabajos.

Algunos datos interesantes a tener en cuenta

- Las unidades de trabajo, son tareas a realizar, consisten de un conjunto de datos de entrada, un programa a ejecutar y una lista de requerimientos que caracterizan a la tarea (cómputo, número para alcanzar el *quorum*, cantidad de tareas a lanzar, memoria, almacenamiento y un tiempo máximo de ejecución).

- Los resultados de cada unidad de trabajo son empaquetados en archivos de salida que luego son enviados al servidor de datos. La arquitectura de BOINC permite que los servidores de datos puedan estar ubicados de forma distribuida, ya que son simples servidores web y no necesitan acceder a la base de datos de BOINC. Los proyectos de BOINC que usan archivos grandes, utilizan servidores de datos replicados y distribuidos, ubicados en instituciones asociadas.

Ventajas e inconvenientes

✓ Cualquier persona, donde quiera que esté, puede contribuir con cualquier proyecto BOINC que haya disponible.

✓ Forma una plataforma de gran potencia a coste cero.

✗ Las aplicaciones tienen que ser compilados mediante unas estrictas librerías de BOINC. Son aplicaciones relativamente homogéneas.

✗ Salvo casos muy puntuales, no es posible el *checkpointing* ni la migración de trabajos.

✗ Como está distribuido por todo internet, donde la seguridad es baja o nula, debe tolerar a los hackers o personas malintencionadas. Para ello debe añadir una redundancia computacional, que le resta mucho rendimiento.

I.3.2. HTCondor

High Throughput Condor, o simplemente HTCondor [B10], es una **plataforma** open source de computación de altas prestaciones especialmente diseñada para aplicaciones CPU-intensivas, donde el tiempo de computación es mucho mayor al de comunicaciones. Se puede usar en un clúster de computadores dedicados (*business grid*) y en computadores "domésticos" (*desktop grid*). Se trata de una plataforma que sigue el paradigma de **computación en intranet**.

A diferencia de la HPC (High Performance Computing) o computación de alto rendimiento, donde prima la potencia de cálculo (FLOPS), la HTC (High Throughput Computing) o computación de alta productividad, a la que pertenece HTCondor, busca la mayor cantidad de trabajos terminados en el menor tiempo posible.

No se trata sólo de un *middleware*, sino de un conjunto de programas, que cada uno tiene una función concreta: Condor, Condor-G, DAGMan, Stork, etc.

Arquitectura

La figura 16 muestra la arquitectura de HTCondor según sus capas de aplicación.

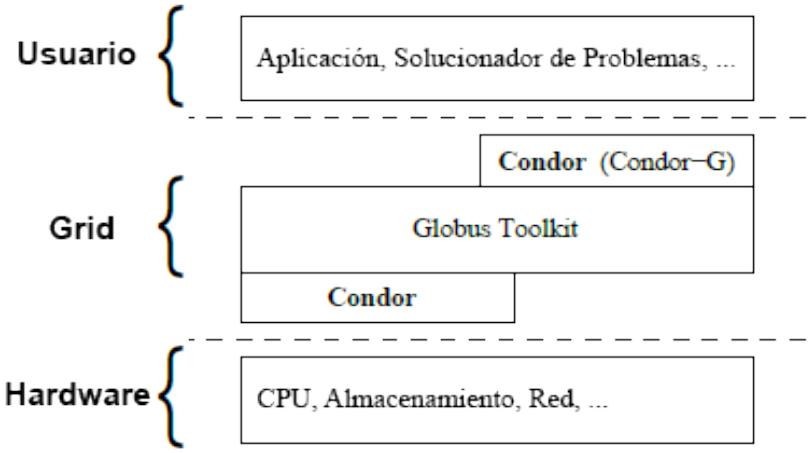


Figura 16: Arquitectura de HTCondor [B10]

- ***Zona Usuario*:** Aquí se encuentran el usuario y las aplicaciones de alto nivel como DAGMan, que controlan la *Zona Grid*.
- ***Zona Grid*:** Es el llamado *middleware*. Conecta las aplicaciones con los recursos. Condor ejecuta aplicaciones en red local, mientras que Condor-G, a través de la herramienta GRAM de Globus Toolkit, conecta con otras plataformas.
- ***Zona Hardware*:** Aquí se encuentran los recursos disponibles: CPU, almacenamiento, red, etc.

Funcionamiento

La figura 17 muestra de forma general las relaciones entre los distintos elementos de HTCondor.

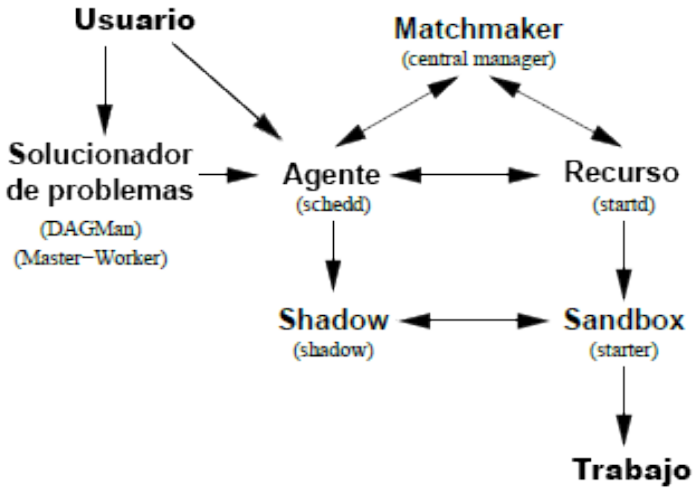


Figura 17: Kernel de HTCondor [B10]

- ***Solucionador de problemas*:** Es una estructura opcional de alto nivel (Master-Worker o DAGMan). Proporciona un modelo de programación para gestionar gran número de trabajos.
- ***Matchmaker*:** Responsable de introducir agentes y recursos potencialmente compatibles.
- ***Agente*:** También llamado *Scheduler*. Responsable de gestionar la cola de trabajos, y de encontrar recursos en los que ejecutarse. Puede ser accedido a través del *Solucionador de Problemas*, o del usuario directamente.
- ***Shadow*:** Representa el usuario para el sistema. Responsable de controlar y monitorizar la ejecución de un trabajo. Hay uno de ellos por cada trabajo en ejecución.
- ***Sandbox*:** Responsable de proporcionar a la tarea un lugar seguro para ejecutarse. Pregunta al *Shadow* los detalles de la tarea, y crea un entorno apropiado a partir de ellos.

Condor-G

Condor-G [B15] es una herramienta cliente que puede gestionar la ejecución de un conjunto de tareas relacionadas en unos recursos computacionales accesibles a través de *grid*. Condor-G permite al usuario especificar el conjunto de tareas a ejecutar, así como las relaciones y dependencias entre ellas. Condor-G proporciona una interfaz cliente agradable para los servicios computacionales Condor y GRAM.

El sistema Condor-G aprovecha los importantes avances que se han logrado en los últimos años en dos áreas distintas: la seguridad, el descubrimiento de recursos y acceso a los recursos en entornos multi-dominio, como apoyo en el globo caja de herramientas, y la gestión de la computación y el aprovechamiento de los recursos dentro de una misma dominio administrativo.

El agente de Condor-G permite al usuario tratar con el entorno *grid* como un recurso enteramente local. Abstrae al usuario de la complejidad. Permite la ejecución remota en recursos gestionados por Globus (Condor, PBS, LSF, etc.).

DAGMan

DAGMan [B1] es un *meta-scheduler* situado a un nivel superior al de Condor-G y Condor. Permite al usuario especificar un DAG (Directed Acyclic Graph, o Grafo Dirigido Acíclico) de tareas con (potencialmente) complejas dependencias y relaciones entre ellas, y que luego sean ejecutadas.

DAGMan optimiza el orden de la ejecución de tareas y trabaja con Condor-G para ejecutar las tareas en la secuencia óptima utilizando los recursos *grid* disponibles.

Los DAGs más complicados son normalmente producidos por programas que más interactúan con el usuario.

Una característica muy interesante es que DAGMan puede distinguir entre programas CPU-intensivos o Data-intensivos. Esto le permitirá mandarlos a Condor/ Condor-G (CPU) o a Stork (datos) según sus necesidades

Los DAGs más complicados se producen típicamente por programas que planifican una serie de tareas basadas en requisitos del usuario (alto nivel). DAGMan proporciona un poderoso sistema de procesamiento de datos [B8].

Stork

Stork [B16] interactúa con planificadores de más alto nivel (DAGMan). Le permite distinguir cuándo un programa es intensivo en CPU o en datos.

También es una herramienta para interactuar con otros recursos heterogéneos. Soporta diferentes sistemas de almacenamiento, protocolos de transporte y middlewares (FTP, GridFTP, HTTP, DiskRouter, SRB, NeST, SRM, etc.).

Utiliza el lenguaje ClassAd para representar los trabajos de colocación de datos [B1].

I.3.3. SZTAKI Desktop Grid

SZTAKI Desktop Grid [B13] es un proyecto de origen húngaro, actualmente gestionado por la IDGF (International Desktop Grid Federation). Consiste en un servidor de BOINC con algunos pequeños cambios, empaquetado en un paquete Debian, para hacer el despliegue lo más fácil posible.

SZTAKI sigue el paradigma de **computación en internet**. Su objetivo es muy similar al de BOINC, atraer el mayor número de voluntarios posible. Aunque dispone también de una versión para intranet, Local SZTAKI Desktop Grid (**computación en intranet**). Básicamente es un servidor BOINC empaquetado en Debian, centrado en aplicaciones locales (empresas o departamentos de universidad por ejemplo). A diferencia de BOINC, SZTAKI no dispone de salvapantallas gráfico. El funcionamiento general es muy similar al de BOINC.

I.3.4. XtremWeb-CH (XWCH)

XtremWeb-CH (XWCH) [W14] es un **middleware** de computación voluntaria que puede desplegar y ejecutar aplicaciones distribuidas y paralelas en infraestructuras de computación. Universidades, centros de investigación y compañías privadas pueden crear su propia infraestructura mientras propietarios anónimos de PCs pueden participar en esas plataformas.

Los voluntarios no se dedican a la informática, a menudo están detrás de firewalls de red que no permite conexiones de red entrantes.

Arquitectura

La figura 18 muestra un esquema general de la arquitectura de XtremWeb-CH

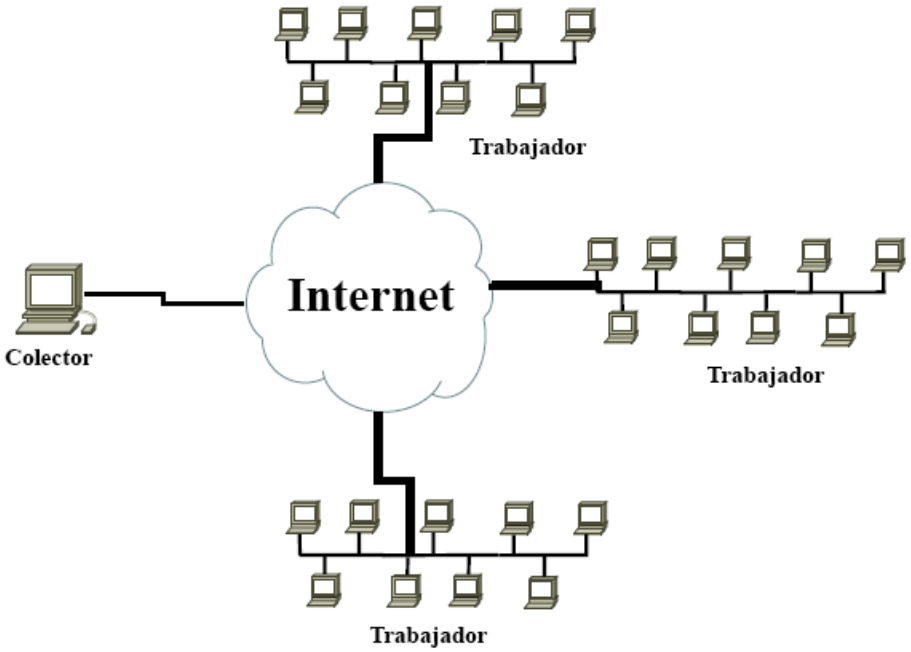


Figura 18: Arquitectura de XtremWeb-CH [W15]

XtremWeb-CH es una versión mejorada de XtremWeb. Permite la ejecución de aplicaciones paralelas/distribuidas compuestas de módulos comunicados.

La comunicación entre los módulos de XtremWeb-CH se hace directamente entre los nodos (trabajadores), sin tener que pasar por el servidor. Esto permite reducir la carga de comunicaciones al servidor. Es una funcionalidad característica del P2P [B14].

Componentes

XWCH consiste en tres componentes principales:

El **colector** acepta peticiones de ejecución de clientes, les asigna tareas y supervisa la ejecución de las mismas.

El **trabajador** recibe la tarea asignada, la ejecuta y, una vez finalizada, devuelve los datos. Funcionan en modo *polling* (encuesta) para evitar problemas de firewalls.

Los **almacenes** actúan como depósito o servidor de entrada/salida de ficheros para las tareas.

Funcionamiento

1. El trabajador recibe una lista de los almacenes disponibles cuando se registra con un coordinador cercano (Solicitud de Registro).

2. Cuando un trabajador envía una solicitud de trabajo para ejecutar una nueva tarea, recibe como respuesta el ejecutable de la tarea asignada y el almacén donde conseguir sus datos de su entrada.

3. Una vez obtenidos el ejecutable y los datos de entrada, el trabajador ejecuta la tarea.

4. Cuando acaba la ejecución de una tarea, carga su resultado en uno de los almacenes conocidos (seleccionado al azar). Por lo tanto, el resultado se almacena en el trabajador y en el almacén.

5. El trabajador envía el resultado del trabajo al coordinador, con las dos ubicaciones de los resultados producidos y se da por concluido el trabajo.

I.3.5. ARC Middleware

El *middleware* ARC (antiguamente NorduGrid) [W30] es una solución software que utiliza tecnología *grid* para permitir compartición y federación de la computación y los recursos de almacenamiento, distribuidos a través de diferentes dominios administrativos y de aplicación. ARC se usa para crear infraestructuras *grid* de diversa envergadura y complejidad, desde un campus hasta *grids* nacionales.

En pocas palabras, ARC se define como:

- Una solución *middleware* de propósito general, open-source, ligera y portable.
- Una implementación de servicios *grid* fiable y escalable.
- Facilitadora de soluciones de computación distribuida entre organizaciones.
- De estándares abiertos e interoperabilidad.
- En continuo uso desde 2002.

I.3.6. gLite

gLite [B17] es un **middleware** para computación *grid* utilizado por el LHC (CERN), para realizar experimentos y otros dominios científicos. Fue implementado conjuntamente por más de 80 personas de 12 diferentes academias y centros de investigación en Europa. gLite proporciona una estructura para construir aplicaciones, aprovechando los recursos de computación y almacenamiento distribuidos por todo Internet. Los servicios de gLite han sido adoptados por más de 250 centros de computación, y usados por más de 15000 investigadores en Europa y el resto del mundo.

I.3.7. UNICORE

UNiform Interface to COmputing Resources [W31]. Es una tecnología de computación *grid* para recursos como supercomputadores o clústeres y para información almacenada en bases de datos.

UNICORE fue desarrollado en dos proyectos fundados por el Ministerio de Educación e Investigación Alemán (BMBF). En proyectos europeos, UNICORE ha evolucionado a un sistema **middleware** usado para centros de supercomputación. UNICORE sirvió como base en otros proyectos de investigación.

UNICORE ofrece un sistema *grid* "preparado-para-correr", incluyendo software cliente y servidor. Sirve para recursos distribuidos tanto en intranets como en todo Internet.

I.3.8. EMI

La Iniciativa Europea de Middleware (siglas EMI en inglés) [W32] no es un programa o middleware como otros, sino un conjunto de middlewares compatibles entre sí. Es el resultado de la unión de tres grandes middlewares de computación, ARC, gLite y UNICORE, y otro especializado en almacenamiento, dCache [W33]. Este último es un sistema para almacenar y recuperar grandes cantidades de datos distribuidas entre un gran número de servidores heterogéneos, mediante métodos de acceso estándar, y bajo un único sistema de ficheros virtual. Anteriormente dichos middlewares funcionaban por separado. EMI comprende un total de 24 socios de distintos países.

En la figura 19 se muestra la unión de los 4 middlewares, y el resultado de la unión, en forma de estándares y servicios.

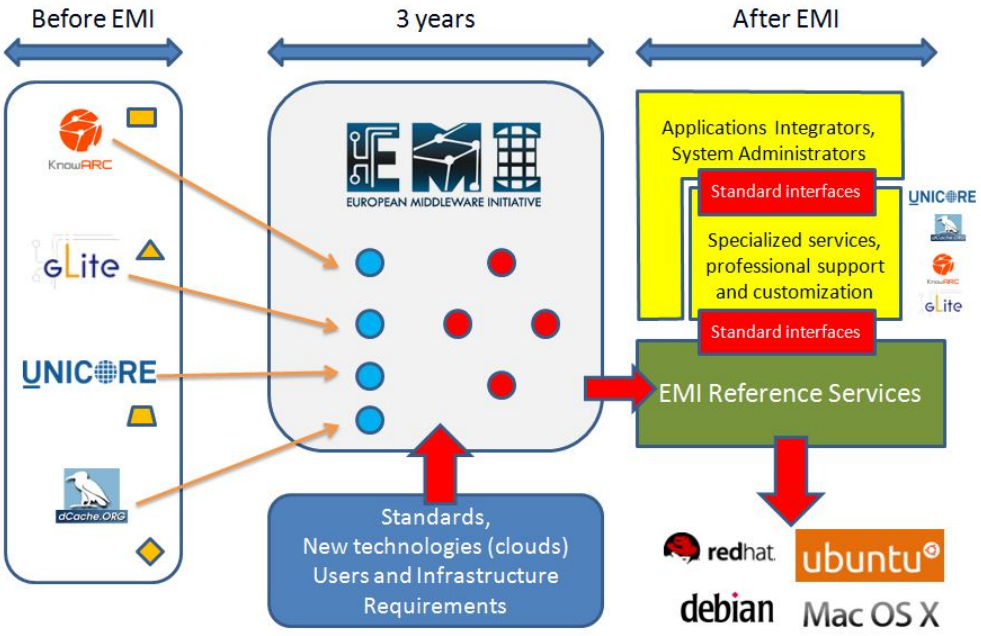


Figura 19: Historia de EMI [W34]

I.3.9. Globus Toolkit

Globus Toolkit [W35] es el principal producto de la Globus Alliance o Proyecto Globus. La Globus Alliance es un conjunto de organizaciones que tienen el objetivo común de construir herramientas que permitan la creación de sistemas Grid. Está formado por instituciones como la Universidad de Chicago, la Universidad de Edimburgo, el Centro Nacional para las Aplicaciones de Supercomputación (NCSA), el Instituto de Ciencias de la Información del Sur de California y empresas como HP, IBM, Intel, Sun, Nortel, Univa y Cisco Systems.

Globus Toolkit es considerado el estándar de facto en la computación grid. Está formado por un conjunto de herramientas desarrolladas bajo licencia open source, componentes software, APIs y librerías que permiten la creación y ejecución de aplicaciones distribuidas, y la construcción de un Grid. La primera versión se publicó en 1998 y la última disponible es la 5.2.5.

Implementa los siguientes estándares: *Open Grid Services Architecture (OGSA), Open Grid Services Infrastructure (OGSI), Web Services Resource Framework (WSRF), Job Submission Description Language (JSDL), Distributed Resource Management Application API (DRMAA), WS-Management, WS-BaseNotification, SOAP, WSDL* y *Grid Security Infrastructure (GSI)* [W36].

Globus consta de tres componentes fundamentales: **gestor de recursos**, **servicio de información** y **gestor de datos** [B18]; todos ellos construidos sobre una infraestructura de seguridad basada en certificados GSI (Grid Security Infrastructure).

- ***Gestión de recursos – GRAM***

La gestión de recursos grid se define como el proceso de identificar requerimientos, hacer matching de recursos con las aplicaciones, asignar estos recursos, planificarlos y monitorizarlos a través del tiempo para lanzar las aplicaciones sobre el grid lo más eficientemente posible.

La arquitectura de gestión de recursos de Globus permite el acceso transparente, unificado y seguro a los distintos gestores de recursos locales de cada centro o institución. Los principales componentes de esta arquitectura son: el lenguaje de especificación de recursos (RSL) y el gestor de asignación de recursos (GRAM).

GRAM (Grid Resource Allocation and Management), proporciona tanto la ejecución remota como su control y monitorización. GRAM proporciona un único interfaz estándar para la ejecución de trabajos, simplificando de esta forma el uso de sistemas remotos.

- ***Servicio de información – MDS***

Los servicios de información constituyen un importante pilar dentro de la infraestructura Grid, ya que proporciona servicios vitales para el descubrimiento y la monitorización, permitiendo de esta manera, planificar y adaptar el comportamiento de las aplicaciones. Este servicio se denomina como MDS (Monitoring and Discovery Service) y proporciona un acceso uniforme, flexible, escalable y eficiente a la información estática y dinámica de recursos, a través del protocolo LDAP.

Los servicios de información se utilizan para indexar, publicar y buscar información relativa a los recursos y servicios disponibles en cada nodo del Grid. Son utilizados por el resto de servicios de Globus que publican información a través de ellos.

- ***Gestor de Datos – GridFTP***

Proporciona transferencias de datos seguras y confiables entre los nodos del Grid. El protocolo GridFTP, se basa en el protocolo estándar FTP (File Transfer Protocol), extendiendo dicho protocolo con otras funcionalidades como transferencias multicanal, auto-ajuste, segmentación de comandos, transferencia entre terceros y seguridad.

GridFTP es una extensión del protocolo FTP. El objetivo de GridFTP es proporcionar una transferencia de ficheros fiable y de alto rendimiento, para permitir la transmisión de ficheros muy grandes. GridFTP se usa en grandes proyectos de la ciencia como el LHC y en centros de supercomputación y otras instalaciones científicas.

- ***Seguridad – GSI***

Los componentes de seguridad de la versión 5 de Globus forman la Infraestructura de Seguridad Grid (GSI, Grid Security Infrastructure). Dicha infraestructura facilita la seguridad de las comunicaciones así como la aplicación de políticas uniformes a través de los distintos sistemas.

GSI ofrece una comunicación segura y autenticada entre los distintos elementos que componen un Grid.

I.3.10. Plataformas HPC de pago

Son plataformas de **intranet computing** similares a HTCondor, pero de carácter propietario y comercial.

I.3.10.1. Grid Engine (Univa)

Grid Engine [W10] corresponde al antiguo SPC de Sun y posteriormente de Oracle. Es el software de computación en intranet más utilizado por empresas y organizaciones.

Se encarga de aceptar, planificar y ejecutar grandes cantidades de trabajos. También se encarga de la gestión y planificación de recursos distribuidos como procesadores, memoria y disco. Se compone de un host maestro y uno o más host de ejecución.

I.3.10.2. LSF (IBM)

LSF (**L**oad **S**haring **F**acility) [W9] es una plataforma propietaria de IBM. Es una ponderosa plataforma de gestión de carga de trabajo para entornos HPC (computación de alta productividad). Proporciona un amplio conjunto de características de planificación, que permite utilizar todos los recursos de infraestructura y garantizar el rendimiento óptimo de las aplicaciones.

I.4. Computación *cloud*

La computación *cloud* es un enfoque nuevo basado en la publicación y consumo de servicios y el aprovisionamiento de recursos de computación a través de Internet, bajo demanda y de una forma flexible, adaptable y altamente configurable.

Las plataformas más comunes son las de infraestructura como servicio o IaaS, plataforma como servicio o PaaS y software como servicio o SaaS. En este proyecto nos centraremos en el análisis del paradigma IaaS, ya que utilizaremos máquinas virtuales para realizar simulaciones.

La principal diferencia la computación *cloud* de la computación *grid* y *cluster* es que en la primera, el usuario no conoce la ubicación real de los recursos computacionales que está utilizando, de forma que no interactúa directamente con los recursos físicos sino que trabaja en una máquina virtual que puede corresponder a cualquiera de los recursos del proveedor de servicios. Esto posibilita un estilo de computación distribuido, flexible y escalable, gestionado de forma transparente al usuario final [B11].

I.4.1. Tipos de *clouds*

Hay 3 tipos de nubes, según su configuración: pública, privada o híbrida.

I.4.1.1. Nube Pública

Es la que se basa en el modelo estándar de cloud computing en el cual los servicios, aplicaciones y almacenamiento se poden a disposición de los usuarios a través de Internet, “como servicio”, normalmente con un modelo de “pague sólo por lo que use”. Existen muchos tipos de nube pública.

Apropiado para usuarios que necesitan infraestructura para solventar necesidades puntuales de computación. **Ejemplos**: Amazon AWS [W17], Windows Azure [W18], Google Compute Engine [W19].

I.4.1.2. Nube Privada

Consiste en una infraestructura *cloud* implantada exclusivamente para una única organización, tanto si se gestiona de forma interna como si un proveedor externo se encarga de ello. Por tanto, constituye a una nueva forma de organizar los recursos propios de una organización sirviendo como alternativa a clústeres o data-centers tradicionales. Las nubes privadas tienen una seguridad avanzada, alta disponibilidad y tolerancia a los fallos que no tienen cabida en la nube pública.

Apropiado para las organizaciones que tienen una gran flexibilidad para hacer balances de carga o para introducir nuevas aplicaciones. **Ejemplos:** Eucalyptus [W20], OpenNebula [W21], Nimbus [W22]

I.4.1.3. Nube Híbrida

Consiste en una combinación de servicios cloud privados (internos) y públicos (externos). Normalmente, las organizaciones ejecutan una aplicación principalmente en la nube privada y utilizan la nube pública de forma excepcional.

La principal utilidad de las nubes híbridas es la de cubrir la demanda de recursos (normalmente de forma puntual) que por diversos motivos no es posible cubrir únicamente con la nube privada. **Ejemplos:** Eucalyptus, Nimbu, OpenNebula.

I.4.2. Arquitectura *cloud*

La figura 20 muestra un ejemplo de arquitectura de un sistema *cloud* típico.

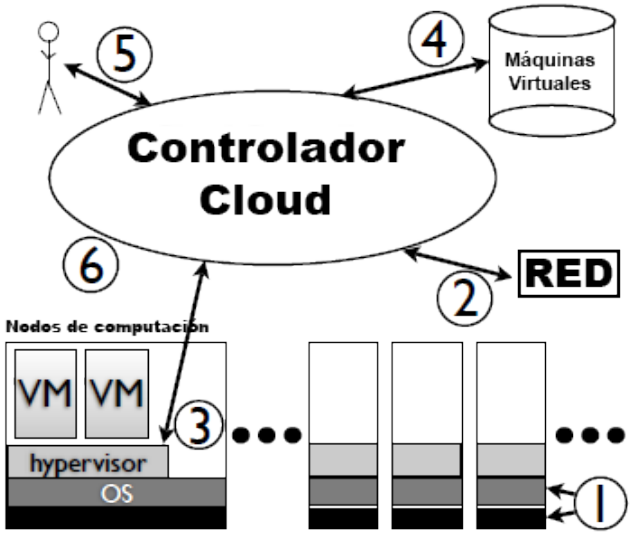


Figura 20: Arquitectura *cloud* [B11]

1. ***Hardware***: recursos físicos (negro) y sistema operativo de las máquinas físicas (gris).
2. ***Red***: conexión física a la red. Incluye los servicios de DHCP y DNS.
3. ***Hipervisor***: software encargado de crear y ejecutar las máquinas virtuales, como si fueran máquinas reales.
4. ***Disco de imágenes virtuales***: aquí se encuentran almacenadas las imágenes de los sistemas operativos que luego se ejecutarán en el hipervisor.
5. ***Usuario final***: el usuario que utiliza las máquinas virtuales.
6. ***Estructura cloud***: el programa que gestionará los recursos de las máquinas físicas, para unificarlas en una única plataforma, a la que accederá el usuario final.

I.4.3. Plataformas *cloud*

Son plataformas que crean una *nube*, es decir, un grupo de ordenadores configurados de tal forma que un usuario final puede solicitar un número de *máquinas virtuales* con una configuración deseada. El usuario final no tiene por qué saber dónde se encuentran físicamente dichas máquinas virtuales, o el hardware que tienen por debajo [B11].

I.4.3.1. Amazon AWS

Amazon Web Services (AWS) [W17] es la plataforma *cloud* más conocida, aunque es de pago. Es una plataforma ya instalada y dispuesta a ofrecer los servicios a cualquiera que los precise, sin preocuparse por el hardware (el hardware corre por cuenta de Amazon). Su filosofía es la de "pague únicamente por lo que necesite".

Cuenta con una gran cantidad de servicios para cubrir todas las necesidades de los usuarios. Dispone de servicios de computación, almacenamiento, bases de datos, análisis y monitorización, etc. De entre todas las funcionalidades de Amazon AWS, nosotros nos centraremos sobre todo en las de computación, ya que es uno de los objetivos del proyecto. Y entre todos los servicios de computación, veremos sobre todo EC2 y VPC.

La función EC2 (Elastic Compute Cloud) ofrece, a través de una interfaz web, máquinas virtuales con una gran variedad de sistemas operativos, a través de la virtualización. También permite personalizarlos, gestionar permisos de acceso a la red y ejecutar tantos sistemas como se desee.

La función VPC (Virtual Private Cloud) trabaja conjuntamente con EC2. Permite la creación de una red privada virtual, en la que luego se configuran las máquinas virtuales creadas con EC2, simulando una red virtual de ordenadores. También es posible unir la red virtual creada con una red real de una organización, mediante VPN (Virtual Private Network).

I.4.3.2. Eucalyptus

Eucalyptus [W20] [B27] es una plataforma open source para implementar una infraestructura de computación *cloud* en clústeres. Es compatible con la interface de computación *cloud* de Amazon EC2, lo que la hace muy atractiva para crear nubes híbridas y para la federación de *clouds*. Está diseñado para ser una alternativa open source del comercial Amazon EC2, especialmente para el entorno empresarial.

Hay una gran separación entre el espacio del usuario y el del administrador. Los usuarios sólo pueden acceder al sistema a través de una interfaz web. El objetivo último es simplificar en la medida de lo posible el acceso a los usuarios, evitando las complejidades. El software se inclina hacia la descentralización de recursos, en la medida de lo posible.

Eucalyptus implementa un sistema de almacenamiento distribuido, diseñado para imitar la plataforma Amazon S3 [W43].

I.4.3.3. OpenNebula

OpenNebula [W21] es un software open source que permite construir cualquier tipo de cloud: privado, público e híbrido. Ha sido diseñado para ser integrado con cualquier tipo de red y almacenamiento, para adaptarse a los recursos existentes. Proporciona soporte para distintos hipervisores (Xen, KVM y VMware ESXi).

OpenNebula tiende a un mayor nivel de centralización y personalización, especialmente para los usuarios finales. La idea de OpenNebula es una nube privada pura, en la cual los usuarios se autentifican y acceden a las funciones de la nube. La centralización supone que su administración es mucho más sencilla.

OpenNebula utiliza por defecto un sistema de ficheros compartidos, como NFS, para todo. Este aspecto puede suponer un problema, pues requiere una gran cantidad de espacio, y puede haber cuellos de botella. Aparte supone un problema de seguridad el hecho de que NFS no encripte el tráfico de datos.

Al ser altamente personalizable, permite combinar la nube privada con otros sistemas como HTCondor. OpenNebula está especialmente dirigido a pequeñas organizaciones, con usuarios experimentados (la personalización puede afectar a la seguridad) que pueden aprovechar las funciones que la plataforma dispone.

I.4.3.4. Nimbus

El proyecto Nimbus [W22] se define propiamente como una solución *cloud* para la ciencia. Pertenece al proyecto Globus.

Al igual que OpenNebula, Nimbus es muy personalizable, pero esta vez es el administrador quien posee únicamente muchas de las opciones disponibles. Es muy flexible en el número y tipo de redes virtuales que pueden ser utilizadas.

Nimbus puede crear una *nube federada*, que podrá interactuar con EC2 u otras plataformas por si hubiese un exceso de demanda. Es un término intermedio entre Eucalyptus y OpenNebula.

Anexo II – Descripción de HTCondor y decisiones de instalación

En este anexo se van a describir las características más importantes de HTCondor [B10], así como las alternativas de configuración que hemos planteado, y las decisiones de diseño e implementación más importantes.

También detallaremos con mayor profundidad el proceso de instalación y configuración de HTCondor, a nivel de implementación. Al final enumeraremos algunas limitaciones del sistema.

II.1. Descripción de HTCondor

En este primer apartado vamos a explicar las características más importantes del sistema. Es decir, los roles que una máquina de HTCondor puede desempeñar, los procesos o demonios que se ejecutan en el sistema, y los diferentes entornos de ejecución.

II.1.1. Roles de máquinas

La plataforma HTCondor [B10] posee una naturaleza distribuida que posibilita la ubicación de diferentes servicios en diferentes máquinas. De esta forma, cada máquina puede tener una sola función o puede tener varias funciones al mismo tiempo. Este apartado describe qué funciones puede desempeñar cada máquina y qué recursos necesita para ello.

- ***Central Manager.*** Es el colector de información y el negociador entre recursos y peticiones o *matchmaker*. Es importante que este servicio esté en una máquina siempre encendida, pues si esta máquina cae, deja de haber *matchmaking*, es decir, se deja de actualizar los recursos disponibles y no se pueden ejecutar nuevas tareas (las ya sometidas siguen ejecutándose). Sólo puede haber un *Central Manager* activo en cada *pool*, aunque puede haber réplicas para mayor fiabilidad. Se compone de 2 demonios, uno para cada tarea: *condor_collector* y *condor_negotiator*.

- ***Execute.*** Ejecuta trabajos sometidos a HTCondor. Cualquier máquina puede hacer esta función, incluso el *Central Manager*. No requiere grandes recursos. Eso sí, cuanto más potente sea la máquina, más peticiones podrá servir. Se compone de los demonios *condor_startd* y *condor_starter*.

- ***Submit.*** Somete trabajos al sistema HTCondor y controla la cola de trabajos. Cualquier máquina puede hacer esta función, incluso el *Central Manager*. Sin embargo, si por alguna razón se cae el demonio que controla la cola de trabajos (*condor_schedd*), pueden perderse los trabajos sometidos. Es pues un punto crítico, debe situarse en una máquina estable. Necesita más recursos que la función "*Execute*" ya que cuanto mayor sea el número de trabajos sometidos, mayor será la cantidad de memoria RAM necesaria para gestionarlos. Cualquier proceso que se está ejecutando en una máquina remota tiene un demonio en la máquina que lo ha sometido (*condor_shadow*). Además, los ficheros de *checkpoint* también

se almacenan en esta máquina, aunque esta última necesidad de disco puede ser aliviada de alguna forma por un *Checkpoint Server*.

- ***Checkpoint Server.*** Es una máquina que almacena ficheros de *checkpoint* de los trabajos. Su único objetivo es el de aliviar en cierta medida al *Submit*, encargándose de la función de *checkpointing*. Esta máquina necesita gran cantidad de disco duro, y una buena conexión a la red (al *pool*), porque el tráfico puede ser muy pesado. Ejecuta el demonio *condor_ckpt_server*. Su uso es opcional. De no usarse, es el *Submit* el encargado de realizar y almacenar los *checkpoints*.

- ***Connection Broker (CCB).*** Es un servicio especial que tiene HTCondor. Tiene dos funciones

    - Es la "cara visible" de HTCondor desde el exterior. Actúa como un proxy entre los demás servicios y las conexiones entrantes, limitando así el número de puertos necesarios a sólo uno.

    - Registra a las máquinas con IP privada no accesible para poder establecer comunicaciones bidireccionales cuando sea necesario.

II.1.2. Demonios

Como se comentó en el apartado anterior, las diferentes funciones que tiene HTCondor son llevadas a cabo por diferentes demonios. Cada demonio es un proceso que se ejecuta en segundo plano y que no tiene interfaz gráfica de usuario, por lo que aparentemente no es visible. Aquí describimos los demonios más importantes que vamos a utilizar.

- **condor_master.** Es el responsable de correr y mantener todos los demás demonios. Se ejecuta en todas las máquinas. Comprueba actualizaciones (y actualiza lo necesario). Si hay errores en algún demonio, avisa al administrador.

- **condor_collector.** Recoge toda la información sobre el estado del *pool* de Condor. Todos los demonios le envían periódicamente actualizaciones con ClassAds (lenguaje utilizado por HTCondor).

- **condor_negotiator.** Es el *matchmaker* , es decir, es el demonio encargado de decidir en qué recurso del *pool* se ejecuta cada trabajo sometido. Periódicamente consulta al colector por el estado actual de los recursos en el *pool*. Todos los demonios le envían periódicamente actualizaciones a través de ClassAds.

- **condor_schedd.** Somete las peticiones al *pool*. Cuando un usuario somete un trabajo, va al *condor_schedd*, donde se almacena una cola de trabajos (administrada por este demonio). También anuncia el número de trabajos en espera en la cola y es responsable de reclamar (al *matchmaker*) recursos para servir estas peticiones. Una vez que un trabajo se enlaza con una máquina ejecutable, crea un *condor_shadow* que sirve esa petición particular.

- **condor_startd.** Prepara la máquina para aceptar peticiones. Se utiliza en la inicialización de los recursos.

- **condor_starter.** Ejecuta y monitoriza el trabajo. Cuando se completa, envía información de estado a la máquina *submit* y termina.

- **condor_shadow.** Se crea cada vez que un trabajo es sometido, para monitorizarlo. Es el encargado de proporcionarle lo necesario a la máquina que lo ejecuta, para que pueda llevarse a cabo sin problemas: ficheros de entrada, posibles *checkpoints*, etc.

- **condor_ckpt_server.** *Checkpoint Server*. Como se ha dicho arriba, es el encargado de realizar y proporcionar *checkpoints*.

- **condor_kbdd.** Monitoriza teclado y ratón, para ver cuándo la máquina está ociosa y cuándo hay alguien utilizándola. Normalmente sólo es necesario para Windows.

- **condor_had.** *High Availability Daemon*. Monitoriza los posibles *Central Manager* para detectar cuándo alguno cae, y sustituirlo.

II.1.3. Universos

En HTCondor, se pueden definir diferentes universos que definan el entorno de ejecución utilizado por cada trabajo sometido. El universo HTCondor es el entorno de ejecución de los trabajos que se ejecutan en las máquinas del *pool*.

Se especifica en el fichero de descripción de *submit* (el que se utiliza para someter los trabajos). Si no se especifica ninguno, el universo por defecto es *vanilla* (a menos que el administrador lo haya cambiado).

- Standard: Proporciona *checkpoints* y llamadas al sistema, pero tiene varias limitaciones: los programas deben ser compilados mediante `condor_compile` con las librerías de HTCondor (necesario el código fuente), no se pueden utilizar llamadas `fork()`, `exec()` o similares. Es el entorno más fiable y eficiente, pero no todos los trabajos lo soportan. La experiencia dice que se utiliza poco debido a las limitaciones que impone.

- Vanilla: Pensado para programas que no pueden ser re-enlazados y para scripts de Shell. No compatible con *checkpoint*. Preferible para sistemas de ficheros compartidos. Es el universo más utilizado de todos, y con el que se ejecutan la mayoría de las aplicaciones.

- Grid: Proporciona una interfaz a usuarios que desean ejecutar sus trabajos en sistemas *grid* remotos.

- Java: Consiste en una máquina virtual Java, con la que un trabajo se puede ejecutar en cualquier máquina, independientemente del hardware o sistema operativo que ésta tenga. Necesario tener la máquina virtual Java previamente instalada.

- Scheduler: Permite a los usuarios enviar trabajos ligeros que se ejecutan de inmediato, en la propia máquina *Submit*. No se utiliza el demonio *condor_starter* para gestionar el trabajo.

- Local: Similar al universo *Scheduler*. La diferencia es que en éste sí se utiliza *condor_starter*.

- Parallel: Proporciona un entorno para programas paralelos, tal que MPI. Antiguamente llamado universo MPI.

- VM: Facilita la ejecución de máquinas virtuales de VMWare y Xen.

II.2. Instalación de HTCondor

En este apartado vamos a explicar el proceso de instalación de HTCondor. Primero detallaremos las decisiones tomadas y sus razones, y después, los pasos a seguir para instalar HTCondor, primero en Linux y finalmente en Windows.

Dado que el proceso de instalación es bien distinto de realizarlo en uno o en otro sistema operativo, lo describimos en apartados separados, aunque una vez instalado, la parte de configuración es muy similar y podemos juntarlo en un mismo apartado.

II.2.1. Consideraciones de seguridad

HTCondor dispone de las funciones de **autenticación**, **encriptación** e **integridad** en el ámbito de la seguridad. De entre ellas, nos va a interesar principalmente la autenticación. Las otras dos no resultan tan críticas por el tipo de entorno en el cual tanto máquinas como usuarios son confiables.

Una de las principales funciones que nos va a interesar del sistema es el hecho de poder someter trabajos de forma remota, a través de la herramienta `condor_submit –remote`. Sin embargo, esta función requiere inevitablemente la autenticación por usuario.

Existen varios métodos de autenticación soportados por HTCondor. Sin embargo, no todos son multiplataforma, ni todos son igual de seguros o complejos de configurar, ni sirven para lo mismo. En la tabla 1 vemos un resumen de las características de los distintos métodos [W23]. En ella mostramos las plataformas compatibles para cada método, el nivel de seguridad que proporcionan (según la capacidad que tienen para evitar intrusiones y situaciones similares, no deseadas), su complejidad a la hora de desplegarlas (evaluado en cuanto a la dificultad para desplegar el método) y los propósitos para los que se pueden utilizar.

	Plataformas	Seguridad	Complejidad	Propósitos
FS	Linux	Alta	Baja	Sólo en local
FS Remote	Linux	Variable	Media	Cualquiera con acceso al NFS
Password	Cualquiera	Alta	Media	Sólo entre demonios
GSI	Linux	Alta	Muy alta	Cualquiera
SSL	Cualquiera	Alta	Alta	Cualquiera
Kerberos	Cualquiera	Alta	Muy alta	Cualquiera
NTSSPI	Windows	Baja	Baja	Cualquiera, sólo autenticación
Claimtobe	Cualquiera	Ninguna	Baja	Debug
Anonymous	Cualquiera	Ninguna	Baja	Debug

Tabla 8: Alternativas de seguridad

- **FS (File System)**: Seguro y fácil de desplegar. Sólo para Linux dentro de la misma máquina.
- **FS Remote:** Seguridad variable, depende del sistema de ficheros compartido. Necesario NFS o similar. Sólo para Linux.
- **Password:** Seguro y de dificultad media. Multiplataforma, aunque sólo para comunicaciones entre demonios. No permite someter trabajos remotamente.
- **GSI (Grid Security Infrastructure):** Seguro pero muy complejo de configurar, y sólo para Linux. Es una herramienta del Globus Toolkit (explicada en el apartado I.3.9. Globus Toolkit) basada en SSL.
- **SSL:** Seguro. Complejo de configurar al principio pero mecánico después. Adecuado para todos los propósitos.
- **Kerberos:** Seguro pero muy complicado para desplegar. Para cualquier propósito.
- **NTSSPI (Security Support Provider Interface):** Sólo para Windows. Seguridad baja, fácil de configurar.
- **Claimtobe:** Inseguro. Cada uno es quien dice ser, se puede engañar fácilmente.
- **Anonymous:** Inseguro. No hay identificación.

Lo más importante que necesitamos es que sea multiplataforma, y que nos permita realizar las funciones que queremos, a ser posible sin una complejidad excesiva. En vista de los resultados, los únicos métodos que permiten someter remotamente y son multiplataforma son SSL y Kerberos. Hemos decidido utilizar **SSL** por ser menos complejo y más conocido. En el apartado II.3.7. Seguridad y autenticación se explica con mayor detalle la configuración y el despliegue.

II.2.2. Otras decisiones importantes

Aquí vamos a describir algunas decisiones de instalación y configuración, que no se engloban en otros apartados, pero no por ello dejan de ser importantes.

- Después de consultar el manual de HTCondor, hemos decidido **no implementar un *Checkpoint Server***, porque no ganamos nada desplegándolo. El único objetivo que tiene es aliviar al *scheduler* asumiendo la tarea de crear y almacenar *checkpoints*. Sin embargo, una de las limitaciones que tiene HTCondor por el momento es que, aunque podamos desplegar varios servidores de *checkpoint*, no es posible configurarlos como espejos de alta disponibilidad. Es decir, únicamente para que los *Executes* accedan al más cercano. Si éste se cae, no contactan con otro. Entonces, si lo ponemos en una máquina estable (la única con la que podemos contar es el *Central Manager*) no alivia al *scheduler*, y si no, la solución final es peor que la inicial [W37]. Además, como se comentó anteriormente, la posibilidad de utilizar checkpoints es muy limitada en la práctica por los requisitos que impone. Asimismo, HTCondor no permite realizar checkpoints de trabajos ejecutados en máquinas *Execute* registradas en el *CCB* (con IP privada fuera de la red del *pool*). Por tanto, esta decisión no tiene un gran impacto en el despliegue del sistema.

- Hemos valorado la opción de configurar espejos o réplicas del *Central Manager*, por si éste fallara, que fuera levantado en otra máquina para seguir funcionando. Finalmente, tras valorar ventajas e inconvenientes, hemos decidido **no implementar la función de *Central Manager* de alta disponibilidad**. HTCondor tiene una limitación aquí: no es posible hacer *flocking* en un sistema con un *Central Manager* con réplicas. Y el hecho de tener que montar las réplicas en máquinas "inestables" (con caídas relativamente frecuentes) hace que nos compensa más configurar la función de *flocking* que ésta.

- Debido a la imposibilidad de crear una carpeta compartida entre el *Central Manager* y todos los posibles *Submit*, **no implementaremos un *Submit* de alta disponibilidad**, aunque cabe la posibilidad de hacerlo en un futuro, si se llegara a instalar HTCondor en alguna máquina con carpetas compartidas comunes como por ejemplo Hendrix.

II.2.3. Instalación en Linux

Hay tres formas de instalar HTCondor en Linux:

1. Compilando el código fuente y distribuyendo los ficheros resultantes.

2. Instalando desde un paquete *.deb* (Ubuntu/Debian/etc.) o *.rpm* (CentOS/Red Hat/etc.)

3. Instalando desde repositorio, a través de *apt-get* (deb) o *yum* (rpm).

La primera opción es la más flexible, pues disponemos de la última versión, que descargamos de la página oficial de HTCondor, además de muchas opciones de instalación. Sin embargo, esta opción presenta muchos problemas tanto a la hora de compilar e instalar (problemas de dependencias, problemas con el compilador, etc.) como de configurar (parámetros mal configurados). Esta opción sólo se recomienda para administradores expertos o para sistemas específicos que necesiten configuraciones muy concretas. Por tanto, en este proyecto, como se busca una solución general y de fácil aplicación, hemos decidido descartar esta opción.

Con la segunda opción también tenemos la última versión, también de su página oficial. Tenemos resuelto el tema de la compilación, pero sigue habiendo problemas de dependencias, por lo que para poder instalar el paquete, antes es necesario instalar todos los paquetes de los que depende.

En la última opción, la versión de HTCondor depende de cuán actualizado esté el repositorio, y suele estar desfasada respecto a las últimas versiones. La principal ventaja es que simplemente introduciendo el comando `sudo yum install condor` en CentOS o `sudo apt-get install condor` en Ubuntu, se instala automáticamente. El instalador resuelve las dependencias y todo automáticamente.

Al final hemos optado por la segunda opción, aunque con ayuda de la tercera. Es decir, al introducir el comando `sudo [yum|apt-get] install condor`, nos indica todos los paquetes que se van a instalar (*condor* y todas sus dependencias). Copiamos entonces todas las dependencias, y las instalamos nosotros por separado. Por último, instalamos el paquete *.deb* o *.rpm* que hemos descargado de la web oficial mediante `dpkg` o `rpm` respectivamente. Por ejemplo, los comandos necesarios en CentOS serían:

```
sudo yum install audit-libs-python cyrus-sasl-md5 gnutls-utils
libcgroup libselinux-python libsemanage-python libtool-ltdl
libvirt-client mailcap perl-Compress-Raw-Zlib perl-Compress-Zlib
perl-Date-Manip perl-HTML-Parser perl-HTML-Tagset perl-IO-
Compress-Base perl-IO-Compress-Zlib perl-Time-HiRes perl-URI
perl-XML-Parser perl-XML-Simple perl-YAML-Syck perl-libwww-perl
policycoreutils-python setools-console setools-libs setools-
libs-python yajl

sudo rpm –i (nombre_paquete_instalación).rpm
```

Y en Ubuntu serían:

```
sudo apt-get install dmtcp libavahi-client3 libavahi-common-data
libavahi-common3 libclassad3 libdate-manip-perl libglobus-
callout0 libglobus-common0 libglobus-ftp-control1 libglobus-
gass-transfer2 libglobus-gram-client3 libglobus-gram-protocol3
libglobus-gsi-callback0 libglobus-gsi-cert-utils0 libglobus-gsi-
credential1 libglobus-gsi-openssl-error0 libglobus-gsi-proxy-
core0 libglobus-gsi-proxy-ssl1 libglobus-gsi-sysconfig1
libglobus-gss-assist3 libglobus-gssapi-error2 libglobus-gssapi-
gsi4 libglobus-io3 libglobus-openssl-module0 libglobus-rsl2
libglobus-xio-gsi-driver0 libglobus-xio0 libgsoap3 libltdl7
libmtcp1 libvirt0 libyajl2

sudo dpkg –i (nombre_paquete_instalación).deb
```

Estos comandos pueden variar ligeramente dependiendo de las librerías instaladas en el Sistema Operativo concreto.

II.2.4. Instalación en Windows

Amazon EC2 [W5] dispone de 3 tipos de instancias Windows gratuitas: Windows 2008, Windows 2008 R2 y Windows 2012. Todas ellas nos pueden servir igualmente, pero utilizaremos Windows 2008 R2, por ser la más parecida a Windows 7 (familiaridad).

Tendremos que descargar el paquete de instalación *.msi* de la página oficial de HTCondor y ejecutarlo.

1. En primer lugar pide el nombre del nuevo *pool*, si vamos a crear uno nuevo, o la IP del *Central Manager* si queremos unirnos a un *pool* existente.

2. Después pide el rol que va a desempeñar la máquina. Le podemos permitir someter trabajos, ejecutarlos, ambas cosas o ninguna (si no es el *Central Manager*, la última opción no tiene mucho sentido). En caso de ejecutarlos, también nos da opción de cuándo hacerlo: siempre, cuando el teclado está 15 minutos inactivo o cuando además también lo está el CPU (podremos personalizarlo más tarde desde el fichero de configuración). Además, en el caso de ejecutar sólo en el tiempo libre, tenemos la opción de guardar trabajos en memoria y reanudarlos después, o de migrarlos a otra máquina. No nos pregunta si nuestra máquina

tendrá el papel de *Central Manager* o no, pues va implícito en el primer paso de la instalación.

3. Introducimos el nombre y DNS de la máquina, nombre@dominio. Esto nos identificará dentro del *pool*. El DNS es opcional, no todas las máquinas lo tienen.

4. Rellenamos el email del administrador y el servidor SMTP donde se enviarán los correos con posibles fallos al administrador. Este paso por ahora no es demasiado importante.

5. Indicamos la ruta de la máquina virtual Java. Si no tenemos instalado Java en nuestro ordenador, la dejamos en blanco. Para poder ejecutar trabajos Java es obligatorio tenerla instalada.

6. Introducimos los permisos. Hay tres tipos: lectura, escritura y administración. Los dos primeros deberían tenerlos todas las máquinas del *pool*. El permiso de administración, sólo el *Central Manager*. Pueden ser cambiados posteriormente.

7. Opcionalmente HTCondor permite ejecutar trabajos en un entorno virtual, a través de VMWare. En ese caso, aquí configuraríamos la memoria disponible, el número máximo de instancias, si hay soporte de red, y la ruta de *Perl*.

8. También de forma opcional podemos configurar HDFS (Hadoop Distributed File System). Requiere Java y Cygwin.

9. Para terminar, especificamos la ruta donde se instalará HTCondor. Normalmente, para facilitar su acceso, se instala en una ruta sencilla como `C:\condor`.

En este caso, tendremos en la misma carpeta, `C:\condor`, todo el contenido del programa: ejecutables, logs, ficheros de configuración, etc. Es pues, una instalación centralizada, a diferencia de la de Linux. El formato de los ficheros de configuración será el mismo. Cambiarán sin embargo, las rutas donde se localicen los ficheros necesarios. Por este motivo, necesitaremos tener dos ficheros distintos de configuración global, *condor_config.* Uno para Linux y otro para Windows. Más luego, cada máquina tendrá su *condor_config.local* particular.

II.3. Configuración de HTCondor

La parte de configuración es relativamente común a Windows y Linux. Aunque por necesidad tengamos un fichero distinto para cada sistema operativo, como son muy similares (cambian las rutas y algunos parámetros menores), vamos a englobar la parte de configuración en un mismo apartado para los dos. El fichero de configuración global es *condor_config*, y está ubicado (por defecto) en `/etc/condor/` en Linux, y en `C:\Condor\etc\` en Windows.

Como fichero de configuración local, tenemos 2 opciones: por un lado podemos tener toda la configuración local concentrada en un fichero en mismo directorio que el fichero global, *condor_config.local*, y por otro, tenemos un directorio, *config.d*, donde podemos crear todos los ficheros que sean necesarios. Cada uno sobrescribirá (si se da el caso de que coinciden variables) al anterior, ordenados por nombre. Se trata únicamente de una cuestión de organización y limpieza. Nosotros hemos optado por el fichero único *condor_config.local*, por considerar más cómodo tener toda la configuración centralizada en un mismo fichero.

Tendremos que mantener eso sí, una buena limpieza, para evitar que se vuelva ilegible (igual que los códigos de programación).

Al arrancar la máquina, primero se inicializan las variables del fichero de configuración global, y después, las del fichero de configuración local, sobre-escribiendo si hubiera conflictos.

Todos los procesos (demonios) de HTCondor se iniciarán como usuario `root` en Linux y `Administrador` en Windows. De esta forma pueden cambiar de propietario fácilmente según el usuario que lo esté utilizando, algo que nos será muy útil.

II.3.1. Carpetas

Directorio **local**, especificado en la variable `LOCAL_DIR`.

- **`execute/`:** Contiene los binarios de los trabajos que se están ejecutando. También contiene los ficheros *core* cuando algún trabajo da error.
- **`spool/`:** Contiene ficheros de colas e historiales de trabajos, y también checkpoints de trabajos de otra máquina (sólo si no se ha desplegado un servidor de *checkpoints*). El espacio depende del número de trabajos y su tamaño. Para la función *High Availability* es necesario que esté en una carpeta compartida.
- **`log/`:** Contiene los *logs* de cada demonio. Es preferible compartir esta carpeta para centralizar los *logs*, aunque no es obligatorio.
- **`lock/`:** Carpeta estrictamente local. Tiene cerrojos o *locks* para acceder a ciertos ficheros compartidos entre varios demonios. Es importante si configuramos los *logs* como compartidos.

Directorio **compartido** o **"release"**, especificado en la variable `RELEASE_DIR`.

- **`bin/`:** Ejecutables de los usuarios finales (en el `PATH`).
- **`sbin/`:** Ejecutables de sistema. Demonios y otros programas que sólo el administrador puede ejecutar (en el `PATH` del administrador).
- **`libexec/`:** Programas sólo ejecutables por HTCondor, no por el usuario.
- **`lib/`:** Contiene librerías, scripts, etc. Tiene permisos para todos (no está en el `PATH`).
- **`etc/`:** Ficheros de configuración, ejemplos y otros.

II.3.2. Configuración global del *pool* de máquinas

Un *pool* o "piscina" es un grupo de máquinas con diversas funciones, similar a un clúster. En este apartado vamos a describir las variables de configuración más importantes. Las variables

que no cambian según el sistema operativo o el entorno se describen de forma concreta, con el valor real. Las que pueden variar se describen con un texto más abstracto.

Lo primero que debemos configurar es lo relativo al *pool* de HTCondor: el **nombre**, la **dirección del colector**, los **permisos** de lectura, escritura y administración, etc. Y también las **rutas** donde se van a ubicar ejecutables, logs, ficheros de configuración, etc.

La variable más importante en un *pool* es la dirección donde estará el colector (la máquina, para localizar el demonio es mediante otra variable):

`CONDOR_HOST =` (IP o nombre de la máquina *Central Manager*)

Después, configuramos las **rutas** de los ficheros:

`RELEASE_DIR =` (directorio “release”)
`LOCAL_DIR =` (directorio local)
`LOCAL_CONFIG_FILE =` (ruta del fichero de configuración global)
`LOCAL_CONFIG_DIR =` (ruta del fichero de configuración local)
`REQUIRE_LOCAL_CONFIG_FILE = TRUE`

La dirección de **correo** del administrador y un programa capaz de enviar correos:

`CONDOR_ADMIN =` (dirección de correo del administrador del sistema)
`MAIL =` (ruta del programa para mandar correo)

Los **dominios de red**, del sistema de ficheros, y la descripción del *pool*:

`UID_DOMAIN =` (sufijo DNS de las máquinas del *pool*)
`FILESYSTEM_DOMAIN =` (sufijo DNS , sólo si utilizamos un sistema de ficheros compartido)
`COLLECTOR_NAME =` (nombre del colector)
`DEFAULT_DOMAIN_NAME =` (sufijo DNS de las máquinas del *pool*)

Los **permisos** más importantes de las máquinas:

`ALLOW_ADMINISTRATOR = $(CONDOR_HOST)`
`ALLOW_OWNER = $(FULL_HOSTNAME), $(ALLOW_ADMINISTRATOR)`
`ALLOW_READ =` (máquinas que pueden ver el estado del *pool*)
`ALLOW_WRITE =` (máquinas que pueden unirse al *pool*)

Además, vamos a utilizar un proxy entre las comunicaciones del *Central Manager* y el exterior (demonio *condor_shared_port*), para que las comunicaciones se establezcan mediante *sockets* (teóricamente ilimitados, salvo por el número máximo de inodos) en vez de mediante puertos TCP (limitados). De esta manera, sólo necesitaremos abrir el puerto 9618 TCP en todas las máquinas, en vez de necesitar pleno acceso entre ellas.

```
USE_SHARED_PORT = True
COLLECTOR_HOST = $(CONDOR_HOST):9618?sock=collector
COLLECTOR_ARGS = -sock collector
NEGOTIATOR_ARGS = -sock negotiator
SCHEDD_ARGS = -sock schedd
STARTD_ARGS = -sock startd
```

```
MASTER_ARGS = -sock master
SHARED_PORT_ARGS = -p 9618
```

Con esta última configuración indicamos que el *condor_shared_port* escuche en el puerto 9618 y que las comunicaciones al colector vayan mediante el *socket* "collector". A otros demonios les renombramos el *socket* para mejorar la legibilidad únicamente. Si no está establecido el nombre del *socket*, se elige aleatoriamente.

II.3.3. Configuración local del *pool*

Como ya hemos visto, existe un fichero de configuración global, igual para todos, y otro específico para cada máquina. Lo que hay en este último añade o sobrescribe lo que haya en el global.

En el fichero de configuración local, tendremos que modificar principalmente, los roles de cada máquina.

En el *Central Manager*:
```
DAEMON_LIST = MASTER, COLLECTOR, NEGOTIATOR, SCHEDD, SHARED_PORT
```

En las máquinas *Execute*:
```
DAEMON_LIST = MASTER, STARTD, SHARED_PORT
```

La configuración específica de cada demonio se define de forma genérica en el fichero de configuración global. Es igual para todos, salvo en algunas excepciones que se explican más abajo: *Submit* de *alta disponibilidad*, máquinas con IP privada no accesible (*Connection Broker* o *CCB*), *flocking* o máquinas ejecutables con una política de ejecución distinta.

II.3.4. Política de ejecución

Debido a que las máquinas que ejecutan trabajos en HTCondor no son máquinas dedicadas, y pueden o no estar disponibles dependiendo del momento, es necesario cierto control. La política de ejecución es la que define cuándo una máquina está disponible para ejecutar trabajos, cuándo deja de estarlo, qué se hace con los trabajos de una máquina que deja de estar disponible y otros casos similares.

Lo que determina cuándo una máquina *Execute* ejecuta un trabajo y cuándo no es la variable `START`. Esta variable del demonio *condor_startd* describe las condiciones que se deben dar en la máquina para poder ejecutar un trabajo. Puede referenciar atributos tanto de máquinas (`KeyboardIdle`, `LoadAvg`) como de trabajos (`Owner`, `Imagesize`). El valor de la expresión `START` juega un papel crucial para determinar el estado y la actividad de una máquina. Pero no sólo tenemos que configurar cuándo se ejecutan los trabajos, también cuándo se suspenden, cuándo se terminan o se matan, cuánto tiempo pueden permanecer en suspensión antes de finalizarlos, etc.

Nosotros la consideraremos ociosa cuando no haya ningún usuario trabajando físicamente con ella (teclado y ratón inactivos), ni ningún usuario remoto esté utilizando de forma significativa

sus recursos (por ejemplo, entrar para consultar un fichero no se considera activa). Es decir, que la CPU no sobrepase un límite establecido.

En Windows, el proceso *condor_kbdd* es el encargado de monitorizar los eventos del teclado y el ratón. Cuando pasa un cierto tiempo sin que se utilicen, se considera que la máquina está libre. En Linux no es necesario, el propio sistema operativo puede hacerlo.

Como extra, podemos priorizar unos trabajos antes que otros, mediante la variable RANK. El uso más común de dicha variable es el de preferencia al propietario o grupo de propietarios de una máquina, antes que el de otros.

La política concreta que se ha configurado en nuestro sistema se muestra a continuación:

- **¿Cuándo se inicia un trabajo en la máquina?** Cuando el teclado está libre y (la CPU está libre o está ejecutando un trabajo con menor prioridad).

```
START = ( (KeyboardIdle > $(StartIdleTime)) && ($(CPUIdle) ||
          (State != "Unclaimed" && State != "Owner")) )
```

- **¿Cuándo se suspende un trabajo?** Cuando se toca el teclado/ratón o cuando la máquina lleva ocupada más de 2 minutos y el trabajo lleva más de 90 segundos ejecutándose.

```
SUSPEND = ( $(KeyboardBusy) || ((CpuBusyTime > 2 * $(MINUTE)) &&
           $(ActivationTimer) > 90 ) )
```

- **¿Cuándo se reanuda un trabajo?** Cuando la CPU y el teclado están libres, y el trabajo lleva al menos 10 segundos suspendido.

```
CONTINUE = ( $(CPUIdle) && ($(ActivityTimer) > 10) &&
           (KeyboardIdle > $(ContinueIdleTime)) )
```

- **¿Cuándo se para un trabajo (ordenado)?** Cuando el trabajo lleva suspendido más de lo que queremos (inicialmente MaxSuspendTime es 10 minutos) o cuando no queremos que se suspenda, pero la máquina deja de estar disponible.

```
PREEMPT = ( ((Activity == “Suspended”) && ($(ActivityTimer) >
$(MaxSuspendTime))) || (SUSPEND && (WANT_SUSPEND == False)) )
```

- **¿Cuándo se mata a un trabajo (brusco)?** En condiciones normales, nunca.

```
KILL = False
```

- **¿Cuándo suspendemos un trabajo en vez de pararlo?** Cuando el trabajo es pequeño, está libre el teclado y el universo es *Vanilla*.

```
WANT_SUSPEND = ( $(SmallJob) || $(KeyboardNotBusy) ||
$(IsVanilla) ) && ( $(SUSPEND) )
```

- **¿Cuándo debemos parar un trabajo en vez de matarlo?** Cuando lleva más de 10 minutos ejecutándose o está en el universo *Vanilla* (en ese caso nunca se mata automáticamente).

```
WANT_VACATE = ( $(ActivationTimer) > 10 * $(MINUTE) ||
$(IsVanilla) )
```

II.3.5. Condor Connection Broker (CCB)

Para poder añadir al *pool* máquinas *Execute* con IP privada, el demonio *condor_shared_port* nos da la posibilidad de actuar como servidor de conexiones (*CCB*). Este demonio registra y mantiene conexiones con las máquinas privadas, para que puedan ser contactadas desde otros demonios.

Para ello, en las máquinas con IP privada no accesible desde el *Central Manager*, añadimos en el fichero de configuración local:

```
CCB_ADDRESS = $(COLLECTOR_HOST)
PRIVATE_NETWORK_NAME = (nombre de la red privada)
```

El nombre de la red privada se usa únicamente para distinguirla en el *CCB*, no tiene por qué coincidir con nombre real (DNS) de la red de la máquina. Sí que es necesario que la dirección del colector sea pública. En Amazon pues, tendremos que cambiar la IP privada fija que ponemos normalmente, por la pública (variable) que nos asignan cada vez.

II.3.6. Alta Disponibilidad (High Availability)

Hay dos tipos de *alta disponibilidad*: la del *Central Manager* y la de la cola de trabajos.

La función *alta disponibilidad* del *Central Manager* nos asegura que si el principal se cae, otra máquina le releva en sus funciones. Sin embargo, es incompatible con la función *flocking*, además de que no disponemos de otras máquinas "estables" (no susceptibles a caídas frecuentes) además de la del *Central Manager* principal, por lo que descartaremos esta configuración.

La *alta disponibilidad* de la cola de trabajos hace que si el *scheduler* actual (el que controla la cola de trabajos) se cae, otro espejo del mismo lo detecta y le sustituye. De esta forma, evitamos que se pierdan los trabajos (aunque haya que reiniciarlos), y permitimos que se puedan someter otros nuevos.

Para que las máquinas *Submit* compartan una única cola de trabajos, tendremos que ubicar la carpeta `spool/` en un directorio compartido, además de añadir en el fichero de configuración local las siguientes expresiones:

```
MASTER_HA_LIST = SCHEDD
SCHEDD_NAME = scheduler@
HA_LOCK_URL = file:$(SPOOL)
VALID_SPOOL_FILES = $(VALID_SPOOL_FILES), SCHEDD.lock
```

Salvo que se someta desde la máquina que tenga el *scheduler* activo en un momento dado, es necesario someter remotamente, y para ello es obligatorio autenticar a los usuarios. Hay varios métodos válidos para tal fin: SSL, Kerberos, FS Remote, etc. Se explica con más detalle en el siguiente apartado.

II.3.7. Seguridad y autenticación

En el apartado II.2.1. Consideraciones de seguridad se enumeraban los distintos métodos de autenticación, con sus pros y sus contras. En dicho análisis se decidió la utilización de SSL, por tanto, en este apartado vamos a detallar su configuración.

Existen tres opciones principales de autenticación: por demonio, por usuario y por host. Nosotros configuraremos el sistema para autenticar por usuario, porque al haber varios usuarios, es la forma más sencilla de controlar los permisos. Se ha diseñado un script que nos facilitará en gran medida el proceso de creación de certificados y claves de usuario, aunque no nos evita el procedimiento inicial, descrito a continuación.

Lo primero que necesitamos es la entidad de certificación o **CA**. Esto es el certificado raíz con el que firmaremos todos los demás que emitamos. Para facilitar las cosas, vamos a disponer de un fichero de configuración personalizado, `openssl.cnf`. Los comandos necesarios para crear la CA son:

`openssl genrsa -des3 -out` **`root-ca.key`** `1024` (introducir un password)

`openssl –in root-ca_crypt.key –out root-ca.key` (el mismo password)

`openssl req -new -x509 -days 3650 -key root-ca.key -out` **`root-ca.crt`** `-config openssl.cnf`

Una vez creada la CA ya podremos crear los certificados de cada usuario (junto con su correspondiente clave privada), y firmarlos con la clave privada de la CA.

Para crear nuevos certificados de usuario se necesitará la clave privada de la CA, que sólo está disponible para el administrador del sistema. Para ello se deberá ejecutar el script **`nuevo_usuario (nombre)`**, que dará como resultado el certificado con la clave pública, `(nombre).crt` y la clave privada, `(nombre).key`.

Es necesario que en el directorio que se ejecute el script se encuentren también los ficheros `root_ca.key`, `openssl.cnf`, `ca.db.serial` y `ca.db.index`.

Cada usuario del sistema con permisos para mandar trabajos, necesitará tener la pareja certificado – clave y el certificado raíz de la CA para verificarlo. Después, el fichero `mapfile` será el encargado de asociar (mapear) el certificado con el usuario real, a través del CN o *Common Name* del certificado.

- **`(nombre).crt`:** Certificado con clave pública. Permiso de lectura para todos.
- **`(nombre).key`:** Clave privada. Permiso de lectura únicamente para el propietario.
- **`root_ca.crt`:** Certificado raíz que verifica los certificados de todos los demás usuarios.

Como restricción, es necesario que los usuarios tengan una cuenta en el *Central Manager* para poder trabajar. De no ser así, los trabajos remotos se someterían como usuario *nobody* y habría muchos problemas para recuperarlos. Si se somete remotamente, el UID:GID del usuario en la máquina remota tiene que coincidir con el existente en el *Central Manager* (fichero

`/etc/passwd`). Crear un nuevo usuario se hace a través del comando `sudo useradd [–u UID] [–g GID] <nombre>`.

II.3.8. Flocking

Flocking es una característica que tiene HTCondor para permitir que dos o más *pools* distintos se junten a través de su colector, compartiendo y complementando los recursos disponibles. De esta forma, cuando se somete un trabajo en un *pool*, y en ese momento no hay ningún recurso capaz de ejecutarlo en ese *pool* pero sí lo hay en otro, ese trabajo se dice que hace *flocking* al otro *pool*, se ejecuta y retorna con los resultados como cualquier otro.

Para poder hacer *flocking* entre dos (o más) *pools*, necesitamos configurar, como mínimo, las siguientes expresiones en el fichero de configuración local del colector (hay más, pero son opcionales):

`FLOCK_TO =` (dirección del colector del otro *pool*. Si el otro utiliza *condor_shared_port* habría que añadir al final `?sock=collector`)

`FLOCK_FROM =` (máquinas que podrán hacer *flocking*. Pueden ser IPs, rangos o sufijos DNS)

Hay un inconveniente: no se aconseja hacer *flocking* a *pools* configurados con un *Central Manager* de alta disponibilidad. En ese caso, como el *scheduler* está configurado con las direcciones de todos los espejos de *Central Manager*, si se añade un colector de otro *pool* distinto, el sistema no lo distingue como debería. En el manual de HTCondor se entra en mayor profundidad [W38].

II.4. Limitaciones

Las limitaciones conocidas que tendremos, tanto por el software de HTCondor como por los recursos de los que disponemos, van a ser expuestas en este apartado.

- **Flocking y *Central Manager* de alta disponibilidad.** Es una limitación de HTCondor. Mientras no se solucione no podremos configurar las dos funciones simultáneamente.

- **CCB y trabajos de universo *standard*.** Lo mismo, según el manual de HTCondor, desde una máquina con IP privada, registrada en el servidor *CCB*, no podremos ejecutar trabajos de dicho universo.

- **Someter remotamente sin tener cuenta en el *Central Manager*.** Esta es una limitación de nuestro sistema. Las máquinas de los laboratorios comparten un mismo espacio de almacenamiento, por lo que existen los mismos usuarios en todas las máquinas. El *Central Manager* sin embargo es una máquina independiente, con sistema de ficheros propio.

  Se llegó a conseguir someter remotamente con un usuario que no existía en el *Central Manager*, pero el trabajo se sometía como usuario *nobody*. Esto tenía dos desventajas importantes: podía generar conflictos entre usuarios y no permitía recuperar los resultados de la cola de trabajos, al no coincidir el usuario real con *nobody*. Estos hechos nos llevaron a

imponer como requisito para someter remotamente el tener una cuenta en el *Central Manager*, con los mismos identificadores (UID y GID) que en las demás máquinas.

- ***Checkpoint Server* de alta disponibilidad.** Como comentábamos en el apartado anterior, el hecho de no poder configurar espejos de un servidor de *checkpoints* es una limitación. Si no configuramos este servidor, la función la realiza el *scheduler*, que sí que puede ser configurado de alta disponibilidad.

Anexo III – Despliegue en Amazon EC2

En este anexo se detallará cómo se ha utilizado la infraestructura de computación *cloud* de Amazon EC2 [W5] para facilitar el despliegue y la configuración de HTCondor [B10]. Primero se analiza las razones por las que hemos elegido Amazon, después describimos el entorno de ejecución que vamos a simular, el despliegue del mismo en Amazon, y para finalizar, analizamos las ventajas que hemos encontrado.

III.1. Motivación y objetivo

El objetivo principal de este proyecto consiste en proponer una solución al problema del uso de recursos efímeros desaprovechados en organizaciones académicas.

En nuestro caso, se plantea la utilización de la plataforma HTCondor en los laboratorios del Departamento de Informática e Ingeniería de Sistemas (DIIS) de la Universidad de Zaragoza [W1].

Debido a la dificultad que lleva hacer cambios en dichos ordenadores, y a la gran cantidad de pruebas necesarias previas al despliegue final, hemos decidido realizarlas en un entorno simulado formado por máquinas virtuales. De esta forma, cualquier posible fallo no comprometería al resto del sistema.

De entre todas las posibilidades, hemos elegido el entorno Amazon EC2 (Elastic Compute Cloud) por la gran cantidad de opciones que nos brinda. Entre ellas están la gran cantidad de sistemas operativos disponibles, la posibilidad de crear una red privada virtual con subredes o la existencia de una capa gratuita, con máquinas poco potentes pero totalmente funcionales.

Por tanto, lo que perseguimos con el uso de las máquinas virtuales de Amazon EC2 es desplegar un sistema de gestión de recursos efímeros simulando un entorno real y evaluando diferentes alternativas de configuración y uso.

III.2. Características del entorno real

En este apartado haremos un análisis de los recursos del Departamento de Informática e Ingeniería de Sistemas. Hemos utilizado este escenario como referencia por ser suficientemente general y representativo, y por la familiaridad que tenemos con el mismo. Analizaremos primero las características hardware y después la configuración de red de los laboratorios.

III.2.1. Descripción de los laboratorios del DIIS

A continuación realizaremos un análisis de los recursos disponibles. La tabla 9 muestra las características principales de los ordenadores del Departamento de Informática e Ingeniería de Sistemas.

	Lab 0.01	Lab 0.02	Lab 0.03	Lab 0.04	Lab 0.05	Lab 1.02
Procesador	Intel Core i5-3470 @ 3.20 GHz	Intel Pentium 4 @ 2.60 GHz	Intel Core 2 Duo E7400 @ 2.80 GHz	Intel Core 2 Duo E7300 @ 2.66 GHz	Intel Core 2 6400 @ 2.13 GHz	Intel Core i5-650 @ 3.20 GHz
RAM	3.39 GB Win 4 GB Linux	1.99 GB	1.94 GB	1.94 GB	1.97 GB	3.42 GB Win 8 GB Linux
HDD	250 GB	500 GB	250 GB	250 GB	320 GB	500 GB
Arquitectura	x86, 32 bits Windows XP Professional x86_64, 64 bits CentOS Linux 6.4 (Kernel en la rama 2.6)					
Red	Intel 82579LM Gigabit Network Connection	NIC Fast Ethernet PCI Familia RTL8139 de Realtek	Intel 82567LM-3 Gigabit Network Connection	Intel 82567LM-3 Gigabit Network Connection	Intel 82566DC Gigabit Network Connection	Broadcom NetXtreme Gigabit Ethernet
Tarjeta gráfica	Intel HD Graphics	Intel 82865G Graphics Controller	Intel 4 Series Internal Chipset	Intel 4 Series Internal Chipset	Intel G965 Express Chipset Family	Intel HD Graphics
Número de ordenadores	30	18	30	30	25	21

Tabla 9: Características de las salas

Todos los ordenadores tienen los sistemas operativos Windows XP Service Pack 3 de 32 bits y CentOS 6.4 de 64 bits con núcleo Linux de la rama 2.6, excepto algunos ordenadores del laboratorio 0.05, que solo tienen Windows XP instalado.

Cabe destacar la diferencia de memoria RAM en los ordenadores que disponen de 4 GB o más. Esto es debido a que el sistema operativo Windows XP dispone de una arquitectura de 32 bits, que lo máximo que admite en teoría son 4 GB, y algo menos en la práctica (3.39 – 3.42 GB). Sin embargo en CentOS, al ser una distribución de 64 bits, se puede aprovechar todo su potencial.

En cuanto al disco duro de cada uno, aquí se muestra la capacidad total, que después de particionar para los demás sistemas operativos, se reduce el espacio aprovechable.

III.2.2. Topología de red de los laboratorios del DIIS

La figura 21 muestra la topología simplificada de los ordenadores del Departamento de Informática e Ingeniería de Sistemas. También se ha añadido el nodo central, ubicado en el I3A (GIDHE) [W2].

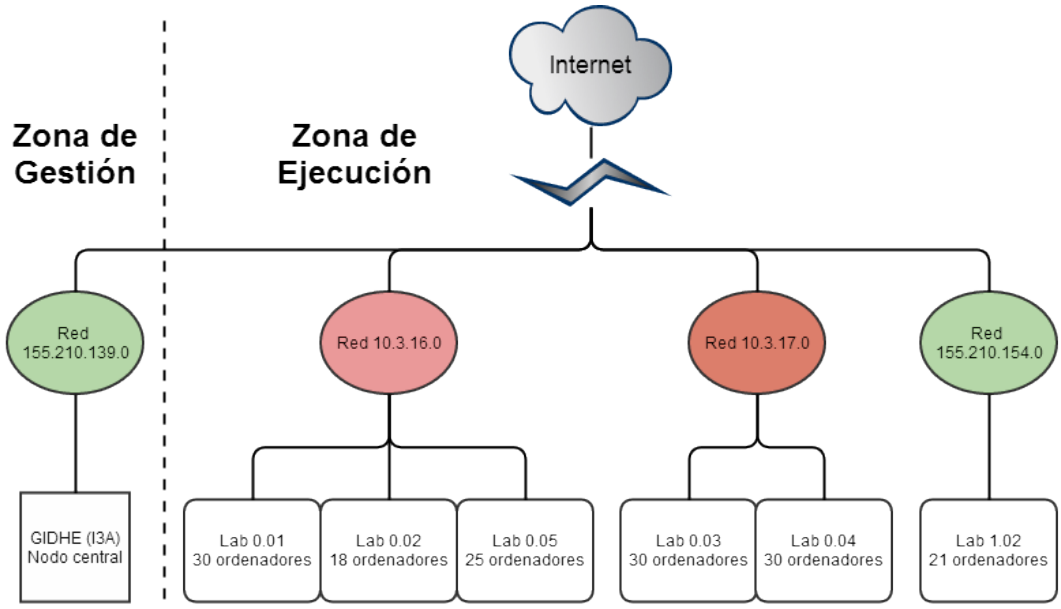


Figura 21: Topología del sistema

Nuestros recursos disponibles inicialmente van a ser los ordenadores de los 6 laboratorios de prácticas. Existe la posibilidad de incorporar nuevos recursos en un futuro, por lo que a la hora de realizar el despliegue, tendremos este hecho en cuenta.

Conceptualmente podemos distinguir tres tipos de máquinas entre todas ellas. Primero tenemos el servidor personal, con IP pública y que además está siempre encendida. Luego tenemos las máquinas del laboratorio 1.02, con IP pública, que además son las más potentes. Y finalmente, las de los demás laboratorios, con IP privada, sólo accesibles desde dentro de la universidad, incluyendo el nodo principal. De todas ellas, sólo el nodo principal será considerado como "estable". Todas las demás máquinas las consideramos como susceptibles a caídas frecuentes, por poder ser utilizados por alumnos en cualquier momento.

- Un ordenador personal en el I3A, con IP pública. Será el único que tengamos con permisos de administrador, y que consideremos "estable", es decir, que no esté sujeto a caídas frecuentes. Lo configuraremos como ***Central Manager***. También será el que controle la cola de trabajos, es decir, el ***scheduler*** (función *Submit*). Y llegado el caso, un servidor ***CCB***, por si queremos incorporar máquinas con IP privada no accesibles desde aquí, para ejecutar trabajos, como hemos comentado arriba.

- Los 21 ordenadores del laboratorio 1.02. Éstos son especiales porque tienen IP pública, y por tanto son accesibles desde todo el mundo, además de ser los más potentes. En un principio los plantearemos como máquinas ***Execute***, aunque dejaremos la posibilidad de que en un futuro sean configurados también como **espejos del *scheduler*** principal.

- Los ordenadores de los demás laboratorios, pertenecientes a las redes privadas 10.3.16.0/24 y 10.3.17.0/24. Un total de 133 ordenadores repartidos en 5 laboratorios (en la práctica habrá menos disponibles al mismo tiempo). Todos ellos los plantearemos como máquinas ***Execute*** únicamente.

III.3. Configuración del entorno

Para realizar el despliegue de HTCondor en Amazon EC2, se ha decidido configurar un entorno similar al de los laboratorios del DIIS de la Universidad de Zaragoza, descrito en el apartado anterior. Las razones para ello son que tenemos un elevado conocimiento de dicho entorno y que el mismo es lo suficientemente general como para aplicarse a cualquier otro entorno de ámbito académico. A continuación, describimos el proceso de configuración del entorno de despliegue.

En primer lugar, creamos una instancia CentOS 6.4 de 64 bits (la que más utilizaremos, por ser el sistema operativo de los ordenadores de los laboratorios), la actualizamos, instalamos el paquete HTCondor y otras herramientas necesarias, y creamos una imagen del disco o AMI. Una vez hecho esto último, podremos crear instancias clones de la imagen de CentOS. Únicamente cambian entre ellas la configuración de red y el nombre de la máquina (basado en la IP). Debido a que la configuración del *Central Manager* cambia bastante respecto a las demás máquinas, tendremos dos imágenes de disco: una para el *Central Manager* (con IP fija y conocida para facilitar su restauración) y otra para los demás. Podríamos crear también otra para las máquinas con Windows, pero como únicamente utilizaremos una o dos para probar, y la instalación es más sencilla, no va a ser necesario.

Para simular la configuración de red de los laboratorios, Amazon nos facilita la función VPC (Virtual Private Cloud) [W27], que permite crear redes virtuales privadas, con varias subredes y una puerta de enlace con salida a internet.

Para ello, hemos creado una red VPC, 10.3.0.0/16. Y dentro de ella, tres subredes: 10.3.16.0/24, 10.3.17.0/24 y 10.3.154.0/24, coincidiendo con las subredes de los laboratorios reales. A cada máquina virtual, además de la IP privada, Amazon EC2 le asigna automáticamente una IP pública dinámica, es decir, que varía cada vez que apagamos y encendemos una máquina.

Una vez creadas las subredes, creamos varias instancias de prueba en varias de ellas. En la red 10.3.154.0 irán el *Central Manager* y las máquinas del laboratorio 1.02 (en la realidad pertenecen a redes distintas, pero al ser públicas son accesibles igualmente). En la red 10.3.16.0 van las de los laboratorios 0.01, 0.02 y 0.05. Y la 10.3.17.0, las de los laboratorios 0.03 y 0.04. Las instancias serán clones de las imágenes que hemos guardado antes (nodo central y nodos ejecutables), y alguna de Windows 2008 R2 para probar la configuración en Windows.

Por último, para simular las restricciones de red (IP privada) y el firewall, hemos creado varios "Grupos de Seguridad". Un grupo de seguridad es un conjunto de reglas de firewall para controlar el acceso a las máquinas virtuales. En total, como conexiones entrantes, permitiremos el acceso al puerto 22 (SSH) en Linux y 3389 (RDP o escritorio remoto) en Windows, y luego en todas, el puerto 9618, que permitirá la comunicación entre las distintas máquinas de HTCondor y todos los paquetes ICMP para poder hacer ping, para pruebas.

El entorno después de todo queda según muestra la figura 22:

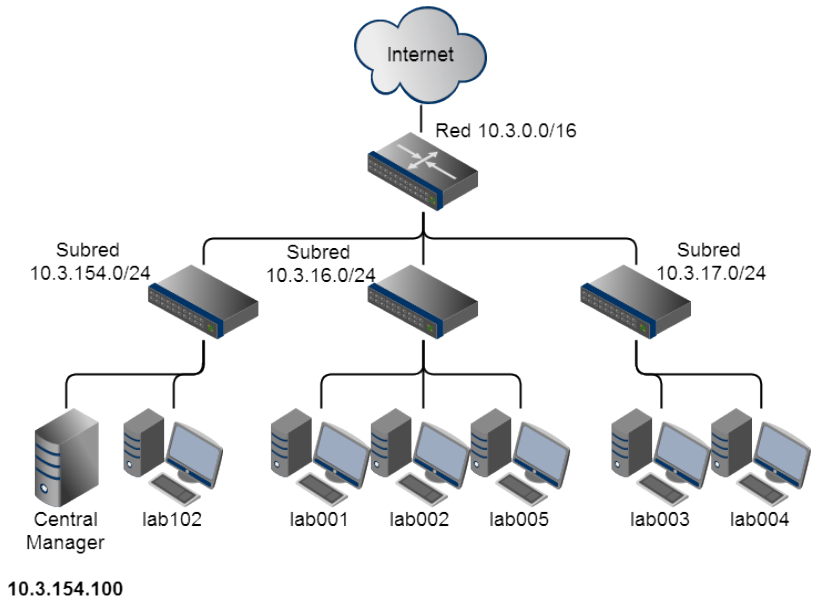


Figura 22: Esquema general de despliegue en Amazon EC2

III.4. Ventajas

El hecho de utilizar un entorno virtual como el que nos proporciona Amazon EC2 para simular un entorno real presenta múltiples ventajas tanto en la referente a la gestión de los propios recursos como a la del software de gestión de recursos efímeros utilizado HTCondor. Esto nos permite probar, eliminar máquinas para volver a crearlas, etc. En general, nos permite cometer errores sin temor a comprometer el resto del sistema.

Por un lado vamos a enumerar las ventajas sobre la gestión de los recursos (las máquinas virtuales) y por otro, sobre la administración de HTCondor, instalación y configuración.

III.3.1. Ventajas de gestión de recursos

En este apartado vamos a enumerar las ventajas que nos supone utilizar Amazon EC2 en lo que a los recursos (las máquinas virtuales) se refiere, respecto al entorno real.

- Nos permite crear configuraciones de red de forma sencilla, con la posibilidad de cambiarlas o eliminarlas en cualquier momento sin esfuerzo.
- Podemos probar situaciones o configuraciones que no podríamos realizar en el entorno real, ya sea por problemas de administración, falta de permisos o falta de recursos.
- Nos proporciona mayor facilidad para simular situaciones de error, y diseñar y evaluar técnicas de recuperación de los errores.
- En general, nos ahorra mucho tiempo y esfuerzo si lo comparamos con el despliegue en máquinas reales.

- Tenemos acceso a todas las máquinas en cualquier momento.

- Una característica muy útil de Amazon es que cada máquina virtual se puede identificar con una IP privada y con otra IP pública, ambas son únicas e identifican a la misma máquina, con la diferencia que la privada sólo es accesible desde dentro de la red VPC mientras que la pública no tiene limitaciones en ese sentido. Gracias a esto, podemos suponer que las máquinas tienen IP pública o privada simplemente utilizando una u otra dirección. Esto simplifica la gestión y facilita el testeo del sistema bajo diferentes condiciones.

III.3.2. Ventajas de administración

Aquí nos referimos a las ventajas relativas a la administración, instalación y configuración de las máquinas virtuales.

- Disponemos de permisos de administrador en todas las máquinas, lo que nos evita todo el proceso necesario para poder hacer cambios en el entorno real.

- Uno de los servicios más importantes que Amazon EC2 proporciona es la posibilidad de poder crear copias exactas del disco duro de una máquina virtual, para poder lanzar máquinas clones de las mismas. Esto nos facilita dos cosas, realizar copias de seguridad y evitar tener que configurar todas las máquinas una por una.

- Al poder restaurar en cualquier momento, podemos realizar un despliegue fácilmente en todas las máquinas sin comprometer el resto del sistema y que afecte a otros usuarios.

- Una característica muy útil de Amazon es que cada máquina virtual se puede identificar con una IP privada y con otra IP pública, ambas son únicas e identifican a la misma máquina, con la diferencia que la privada sólo es accesible desde dentro de la red VPC mientras que la pública no tiene limitaciones en ese sentido. Gracias a esto, podemos suponer que las máquinas tienen IP pública o privada simplemente utilizando una u otra dirección. Esto simplifica la gestión y facilita el testeo del sistema bajo diferentes condiciones.

Anexo IV – Pruebas de uso

En este anexo describimos todas las pruebas sobre el sistema. También vemos los escenarios y problemas que nos podemos encontrar, con alguna posible solución. Realizaremos pruebas tanto de funcionamiento general como de errores. Y finalmente haremos una simulación con la configuración tal y como se plantea para instalar en el entorno real de los laboratorios del Departamento de Informática e Ingeniería de Sistemas (DIIS) de la Universidad de Zaragoza [W1]. El objetivo es comprobar el correcto funcionamiento del sistema, detectar posibles errores y solucionarlos.

Las pruebas han sido automatizadas para ser ejecutadas independientemente del entorno de despliegue (Amazon, laboratorios u otro entorno distinto de utilización).

IV.1. Escenarios posibles

En este apartado veremos los distintos tipos de fallos, lo que le sucede a cada máquina con cada uno, y otros escenarios importantes que nos pueden suceder.

IV.1.1. Escenario de caídas

Hay tres tipos de caídas, desde la visión de HTCondor [B10]:

4. El sistema se vuelve inalcanzable temporalmente. Esto es, que a un ordenador que estaba libre, llega alguien que lo va a utilizar. Cuando termina, bien lo deja como estaba, o bien lo apaga.

5. Apagado ordenado. Apagado correcto de la máquina, ejecutando los scripts de apagado, entre los que se encuentra uno de HTCondor, que notifica al colector que la máquina deja de estar disponible de forma indefinida. También es equivalente el parar un demonio de forma ordenada mediante el comando `condor_off`.

6. Apagado brusco. Se da cuando el sistema se cuelga, cuando alguien apaga el ordenador desde el botón de encendido, quitando el enchufe, cuando se va la luz, etc. Es equivalente a matar el proceso mediante el comando `kill -9` o `condor_off –fast`.

Ahora analizaremos lo que le ocurre a cada una de las máquinas y a la cola de trabajos, con cada tipo de apagado.

Central Manager

Al ser la máquina principal, está siempre operativa, haya alguien trabajando en el equipo o no. Por tanto, no puede volverse inalcanzable temporalmente. Asimismo, esta máquina no debería apagarse de forma ordenada a no ser que quiera pararse todo el sistema HTCondor.

Si el *Central Manager* cae, se deja de hacer *matchmaking*. Las tareas pendientes se terminan de ejecutar sin problema. Las que están en cola, permanecen en ella sin problemas, pero no pueden ser emparejadas con una máquina para ser ejecutadas hasta que el negociador vuelva a estar disponible. Se pueden someter nuevas tareas desde otra máquina *Submit* operativa (si la hay), pero ocurre lo mismo que con las tareas encoladas, que no se emparejan. Tampoco se recoge información sobre el estado del *pool*, pues el colector que es el encargado de hacerlo, no está disponible.

Si esto ocurre, poco se puede hacer más allá de reiniciar el servicio o la máquina. Como mucho se podría notificar al administrador para acelerar el proceso.

Una posible **solución** a este caso sería utilizar la función de *alta disponibilidad*. Se pueden configurar varias máquinas que pueden actuar como espejos (*mirrors*) del *Central Manager* si el principal cayera. Todas estas máquinas estarían en espera (o realizando otras funciones) con el demonio *condor_had* monitorizando conexiones entre unas y otras. En cuanto detectan que el principal falla, se ponen de acuerdo entre las demás (mediante protocolo) para sucederle. En tal caso, todas las máquinas *Submit* y *Execute* se configuran para contactar con cualquiera de los posibles espejos del *Central Manager*.

Estos espejos deben estar en máquinas (relativamente) estables. Como en nuestro caso no se da esa situación, no vamos a configurar esta función porque apenas ganamos nada. Sí que dejamos la posibilidad de hacerlo en un futuro. En el apartado II.2.2. Otras decisiones importantes entramos más en detalle en este tema.

Submit

El *scheduler* (demonio *condor_schedd*) es el encargado de gestionar la cola de trabajos y de crear un demonio *condor_shadow* por cada trabajo que se esté ejecutando. Es necesario que entre cada *condor_shadow* y el *scheduler* haya una comunicación bidireccional. Si esta comunicación se pierde, los trabajos que acaben no podrán mandar de vuelta el resultado en ese periodo. Y si ese tiempo excede un *timeout* establecido, los trabajos se pierden y habría que someterlos de nuevo.

Si perdemos el tiempo invertido en ejecutar los trabajos hasta entonces es malo, pero dentro de lo que cabe, soportable. Si perdemos la cola de trabajos hay un desastre, más aún si la que se pierde es la única que hay. Para evitar esto último, la cola se almacena de manera persistente en disco, en el fichero `job_queue.log`. De esta forma, se podría recuperar una cola válida y restaurar los trabajos que se habían sometido. En el peor de los casos (si hay un apagado brusco), la cola podría corromperse. En este caso poco se puede hacer más allá de restaurarla de una copia de seguridad anterior.

La mejor **solución** es configurar un *Submit* de alta disponibilidad (*High Availability Submit*). En ese caso es necesario un sistema de ficheros compartido, o como mínimo, una carpeta compartida donde localizar la cola de trabajos y un cerrojo para proporcionar acceso concurrente a la misma. Las máquinas hacen de espejos del *scheduler*, comparten una misma cola de trabajos. Con esta solución, cuando una máquina *Submit* cae, dejan de ejecutarse los trabajos que ésta había sometido, migran a otra máquina activa y se reinician (desde cero o desde un *checkpoint* reciente). También se admiten nuevos trabajos. En el caso, menos

probable, de que ninguno de los espejos *Submit* estuviese disponible, el problema es el mismo que arriba.

Execute

El comportamiento de las máquinas *Execute* con los trabajos y las caídas se define en la política de ejecución, en el apartado II.3.4. Política de ejecución. Dependiendo de cómo la hayamos configurado, actuará de una manera u otra.

En nuestro caso, cuando una máquina que está disponible y ejecutando un trabajo deja de estarlo, se mata al trabajo (se le permite hacer un *checkpoint* si lo soporta) y vuelve a la cola para migrar a otra máquina. No lo dejamos en suspensión porque a priori no sabemos cuánto tiempo va a estar la máquina ocupada, y es muy probable que el tiempo de espera sea mayor al de re-ejecución en otra máquina disponible.

Cuando la máquina se apaga ordenadamente, el resultado es muy similar: se ejecuta un script que mata a los trabajos en ejecución, con la posibilidad de crear un *checkpoint* en los trabajos en ejecución que lo soportan (los del universo *standard*). Acto seguido, dichos trabajos migran a otra máquina, y comienzan de cero o de un *checkpoint* reciente.

Cuando se apaga bruscamente, los trabajos en ejecución se pierden en cualquier caso, pues no da tiempo a realizar *checkpoint*. En ese caso, la máquina que ha sometido el trabajo que acaba de perder, debe detectarlo y lanzarlo en otra máquina disponible. Tanto en este caso como en el anterior no hay alternativa posible.

Checkpoint Server

Lo ideal es que los *Checkpoint Servers* estén en una máquina muy estable. Si se cuelga un servidor de *checkpoint*, el sistema HTCondor continuará funcionando, pero pobremente. Hay dos problemas que pueden ocurrir mientras éste se recupera:

1. No se pueden enviar nuevos *checkpoints*. Los trabajos seguirán intentando contactar el servidor de *checkpoint*, con un intervalo de tiempo que se duplica cada vez. Normalmente los trabajos tienen un tiempo limitado para hacer *checkpoint* antes de ser expulsados de la máquina. Si el servidor de *checkpoint* está inactivo demasiado tiempo, lo más probable es que gran parte del trabajo se pierda.

2. No se pueden recuperar trabajos desde un *checkpoint*. En tal caso se reinician desde cero, o bien se quedan a la espera de que el servidor vuelva a estar activo. Esto se controla a través de la variable `MAX_DISCARDED_RUN_TIME`, que representa el tiempo máximo de CPU que un trabajo esperará al servidor *checkpoint* antes de descartarlo y empezar de cero.

Es posible configurar varios servidores *checkpoint* en un mismo *pool*, pero no como espejos que aumenten la fiabilidad, sino para que cada máquina acceda al más cercano. En el apartado I.7. Otras decisiones importantes se explica por qué no lo vamos a utilizar ahora, aunque lo planteamos por si en el futuro se decidiese desplegarlo.

Connection Broker (condor_shared_port)

Cuando una máquina está configurada para utilizar el o *shared port* (mediante la variable `USE_SHARED_PORT = True`), todas las comunicaciones entre demonios y entre otras máquinas pasan por el demonio *condor_shared_port*, que actúa como *proxy*. El tal caso, es el único demonio que escucha en un puerto TCP, el único punto visible desde el exterior. Los demás se comunican entre ellos mediante *sockets*.

El *shared port* se trata de forma especial, porque al ser el único punto visible desde el exterior, si se cae, es como si se cayera toda la máquina, pues se queda incomunicada. Cuando se ejecuta `condor_off` el demonio *condor_shared_port* se deja activo, para prevenir al *condor_master* de un estado en el que no pueda recibir más órdenes.

Si la comunicación entre *Central Manager* y demás máquinas se realiza a través del CCB (*condor_shared_port*), el hecho de que se caiga este servidor es muy similar al de que se caiga el *Central Manager* (colector + negociador), puesto que deja de haber comunicación con ellos. Desde el punto de vista de las demás máquinas, es como si no estuvieran.

Es crítico que el servidor CCB o *shared port* permanezca activo siempre junto al *Central Manager*. Para reducir la posibilidad de caída, es posible especificar múltiples direcciones de CCB (variable `CCB_ADDRESS`), a modo de redundancia.

Cualquier demonio individual

Cuando alguno de los demonios de la máquina se cuelga (cualquiera menos el *condor_master*), es posible que la máquina sigue operativa. En ese caso el demonio *condor_master* manda un correo a la dirección especificada en la variable `CONDOR_ADMIN` con las últimas líneas del log del demonio que ha fallado y con la señal o el estatus de salida devuelto por este. También restaura el demonio si es posible.

El administrador del sistema HTCondor tiene la opción de iniciar, parar o reconfigurar remotamente cualquiera de ellos, a través del *condor_master*. Cuando el proceso que falla es *condor_master*, al ser el padre/propietario de todos los demás procesos de HTCondor, hace que todos los demás también fallen. En ese caso la única posibilidad es volver a iniciar el servicio: `service condor (re)start`.

Resumen

La tabla 10 resume los escenarios de fallo que pueden aparecer en el sistema, explicados con más detalle en este mismo apartado, arriba. Para cada posible fallo se indican dos casillas, la primera indica el problema y la segunda, una posible solución.

	Cambio de libre a ocupado	Apagado ordenado	Apagado brusco
Central Manager	No le afecta.	Se notifica al admin y se deja de hacer *matchmaking*.	Se deja de hacer *matchmaking*.
	–	Configurar espejos para poder restaurarlo en otra máquina. *Alta disponibilidad*.	Configurar espejos para poder restaurarlo en otra máquina. *Alta disponibilidad*.
CCB Shared Port	No le afecta.	La máquina queda incomunicada, como si no estuviera.	La máquina queda incomunicada, como si no estuviera.
	–	Restaurar el demonio *condor_shared_port*	Restaurar el demonio *condor_shared_port*
Submit	No le afecta.	Los trabajos en ejecución terminan y esperan, pudiendo llegar a perderse.	Los trabajos en ejecución terminan y esperan, pudiendo llegar a perderse. Puede corromperse la cola.
	–	Levantar de nuevo el demonio *condor_schedd*	Levantar de nuevo el demonio *condor_schedd*
Submit con espejos (*High Av.*)	No le afecta	Los trabajos en ejecución migran a otro *Submit* espejo.	Los trabajos en ejecución migran a otro *Submit* espejo. Puede corromperse la cola.
	–	Configurar espejos “estables” para reducir este caso.	Configurar espejos “estables” para reducir este caso.
Execute	Se matan los trabajos que estuvieran ejecutando.	Migración, comenzando de 0 o de *checkpoint* reciente.	Migración desde 0
	Cambiar la política de ejecución, o mejorar el algoritmo de *scheduling*.	Pocas soluciones posibles. Encender la máquina o la plataforma HTCondor.	Pocas soluciones posibles. Encender la máquina o la plataforma HTCondor.
Checkpoint Server	No le afecta.	No se crean *checkpoints* ni se restauran trabajos desde *checkpoint*.	No se crean *checkpoints* ni se restauran trabajos desde *checkpoint*.
	–	Restaurar el demonio *condor_ckpt_server*. No es posible utilizar espejos.	Restaurar el demonio *condor_ckpt_server*. No es posible utilizar espejos.

Tabla 10: Escenarios de caídas

IV.1.2. Otros posibles escenarios

Además de los escenarios de caídas explicados en el apartado anterior, también hemos analizado otros posibles escenarios de uso, de cara a proporcionar una solución general para el uso de HTCondor como gestor de una infraestructura de recursos efímeros. A continuación los detallamos.

- **Añadimos nuevos recursos de IP pública**

El añadir recursos con IP pública no supone grandes problemas, pues son accesibles desde todo el mundo. Lo que tenemos que hacer es configurar las nuevas máquinas para que accedan al colector de nuestro *pool*, y darles permisos de lectura y escritura para que se puedan unir a él.

`CONDOR_HOST =` (IP/DNS del *Central Manager*)
`ALLOW_READ =` (@ nuevas máquinas)
`ALLOW_WRITE =` (@ nuevas máquinas)

Otra opción es teniendo dos *pools* distintos, hacer *flocking*. Es decir, unir los dos *pools* en caso de necesidad (de normal solo se usan recursos del propio *pool*). En el apartado II.3.8. Flocking se explica con más detalle esta posibilidad.

- **Añadimos nuevos ordenadores con firewall y/o IP privada**

Los firewalls y las redes privadas o NAT suponen un problema para HTCondor. Hay dos formas de solucionar el primer problema, aunque sólo una nos servirá también para el segundo.

La primera solución consiste en restringir el número de puertos que habría que abrir en el firewall, dejando sólo los necesarios. Y luego el demonio *condor_shared_port* haría de intermediario, gestionando todas las conexiones de los demás demonios.

La segunda solución es usar el *HTCondor Connection Brokering* (CCB). Es un servicio externo que se le otorga al colector (*Central Manager*). Es necesario pues, que el servidor CCB sea accesible tanto por las máquinas de la red privada como por el colector (IP pública). El proceso de conexión es el siguiente:

1. El nodo *Execute* se registra en el servidor CCB (variable `CCB_ADDRESS`).
2. El nodo *Submit* le pide al CCB que el nodo *Execute* abra una conexión TCP.
3. El CCB le remite al *Execute* la petición del *Submit*.
4. El nodo *Execute* abre la conexión TCP. Una vez abierta la conexión, ya es posible la comunicación bidireccional.

En la situación inversa, *Submit* privado y *Execute* público, se aplicaría la misma idea pero al revés.

Como inconveniente, CCB no soporta trabajos del universo *Standard*.

`CCB_ADDRESS = $(COLLECTOR_HOST)`
`PRIVATE_NETWORK_NAME =` (nombre de la red privada)

Si ambas partes estuvieran en una red privada (no accesibles entre ellas), la configuración es más compleja. Sería necesario hacer redirección de puertos (*port-forwarding*) [W39]. En nuestro sistema no se va a dar este caso.

- **Diferencias entre tener *alta disponibilidad* y no.**

Existe un recurso en HTCondor llamado *High Availability* o *alta disponibilidad*, que permite minimizar el problema causado si se cae el *Central Manager*, o una máquina *Submit* con su correspondiente cola de trabajos pendientes.

Central Manager: (Sección 3.11.2)

En vez de una, elegimos varias máquinas que pueden ser posibles *Central Manager*. Todas ellas salvo una permanecen en espera, hasta que la principal deja de estar disponible por alguna razón. Entonces una de ellas le sucede. Todas las máquinas del *pool* están configuradas con las direcciones de todos los posibles *Central Managers*.

Si esta opción está habilitada, si el *Central Manager* principal se cae, inmediatamente se restaura en otra máquina, por lo que la posibilidad real de caída del sistema se minimiza drásticamente. Sin embargo, esta función es incompatible con el recurso *flocking* (más información en II.4. Limitaciones y II.3.8. Flocking)

Cola de trabajos: (Sección 3.11.1)

Necesitamos que la carpeta `spool/` (donde se encuentra la cola de trabajos) sea compartida entre todas las máquinas *Submit* espejo, pues todas ellas comparten una única cola de trabajos. También es necesario un sistema de acceso concurrente: cerrojos o *locks* (fichero `SCHEDD.lock`).

Lo que nos permite esta opción es evitar que cuando una máquina *Submit* caiga, se pierdan los trabajos que ha sometido, y se dejen de aceptar otros nuevos. Es decir, si se cae el *scheduler*, se levanta una réplica en otra máquina con la misma función.

El problema en nuestro sistema es que si se cae el *Submit*, es muy probable que también caiga el *Central Manager*, al estar en la misma máquina. En ese caso, aunque se habilitara un *Submit* espejo en otra máquina, si no se restablece otro espejo del *Central Manager* también (lo cual tiene limitaciones), la ganancia no es mucha, pues el sistema sigue sin funcionar como debería.

IV.2. Programas de prueba

En este apartado vamos a mostrar los programas que hemos utilizado para realizar las pruebas. Son dos: uno en C para probar los universos *Vanilla* y *Standard*, con pequeñas diferencias entre la versión de Windows y la de Linux, y otro en Java, para probar el universo homónimo.

IV.2.1. Programas C

Para probar el funcionamiento general del sistema, hemos creado dos programas de prueba escritos en C, uno para Linux y otro para Windows. Dichos programas funcionan exactamente igual: tienen un argumento que indica el tiempo en segundos que durará su ejecución. En ese tiempo, el programa ejecuta un `sleep()`. Aunque la carga de CPU sea nula, para HTCondor cuenta como que está en ejecución, que es lo importante.

```
#include <stdio.h>
/* #include <windows.h> */

main(int argc, char **argv) {
   int sleep_time;
   int error;

   if (argc != 2) {
      printf("Uso: simple <tiempo_sleep>\n");
      error = 1;
   }
   else {
      sleep_time = atoi(argv[1]);

      printf("Durmiendo durante %d segundos...\n", sleep_time);
      sleep(sleep_time); /* en Linux */
      /* Sleep(sleep_time*1000); en Windows */
      printf("Despertando, %d segundos despues\n", sleep_time);
      error = 0;
   }
   return error;
}
```

IV.2.2. Programa Java

En este caso el mismo programa nos servirá tanto en Windows como en Linux. El programa tomará como entrada un fichero, lo copiará y lo guardará en otro fichero con distinto nombre.

```
import java.io.*;

public class CopiaFicheros {
   public static void main(String[] args) throws IOException {
      FileReader entrada = new FileReader(new File(args[0]));
      FileWriter salida = new FileWriter(new File(args[1]));
      int dato;

      while ( (dato = entrada.read()) != -1 ) {
         salida.write(dato);
      }
      entrada.close();
      salida.close();
   }
}
```

IV.3. Casos de prueba

Para realizar las pruebas hemos utilizado las máquinas virtuales de Amazon EC2, con sistemas operativos CentOS y Windows Server 2008 R2. Adicionalmente, disponemos de un ordenador personal de un laboratorio del I3A, con IP pública de la universidad y sistemas operativo Windows 7 y Debian, y también un portátil personal de casa, con IP privada y sistemas operativos Windows 7 y Linux Mint. El entorno desplegado en Amazon EC2 se describe con mayor detalle en el Anexo III – Despliegue en Amazon EC2.

Cada prueba se describe en una tabla diferente. Cada tabla se compone de lo siguiente:

- **Título:** El nombre de la prueba.
- **Requisitos:** Parámetros que se deben cumplir para el correcto desarrollo de la prueba.
- **Objetivo:** Lo que pretendemos comprobar con la realización de la prueba.
- **Desarrollo:** La parte principal. Describe cómo transcurre la prueba y cómo se comprueba al final si el resultado es satisfactorio o no.
- **Resultado:** Describe el resultado final de la prueba, si es lo que esperábamos o no.

V.3.1. Casos de prueba de funcionamiento general

En estos casos lo que vamos a perseguir es ver que el sistema funciona correctamente en un entorno “normal”. El último caso va a ser una prueba en condiciones reales. Esto es, una prueba

general para comprobar el correcto funcionamiento del sistema. En el caso de Amazon EC2, se ha utilizado para los experimentos una configuración que simula los laboratorios, de acuerdo a lo descrito en el Anexo III – Instalación y configuración de HTCondor. Las siguientes tablas muestran todas las pruebas de funcionamiento general que hemos realizado.

Título	**Someter localmente en Linux**
Requisitos	*Central Manager* con función *Submit* funcionando, una o más máquinas *Execute*. Un usuario sin permisos de *root* con cuenta en el *Central manager* y certificado/clave SSL.
Objetivo	Someter un trabajo desde el *Central Manager* y ver qué ocurre con el trabajo.
Desarrollo	El usuario se conecta al *Central Manager* y ejecuta el comando `condor_submit (nombre_fich.sub)`. Espera a que termine su ejecución (puede comprobarlo mediante `condor_q`) y comprueba que el sistema le ha devuelto los resultados y el log correctamente.
Resultado	Satisfactorio. El programa se ha ejecutado y ha devuelto automáticamente los resultados al directorio señalado.

Tabla 11: Prueba someter localmente en Linux

Título	**Someter localmente en Linux, universo Java**
Requisitos	*Central Manager* con la máquina virtual Java instalada y con función *Submit* funcionando, una o más máquinas *Execute*. Un usuario sin permisos de *root* con cuenta en el *Central manager* y certificado/clave SSL.
Objetivo	Someter un trabajo Java desde el *Central Manager* y ver qué ocurre con el trabajo.
Desarrollo	El usuario se conecta al *Central Manager* y ejecuta el comando `condor_submit (nombre_fich.sub)`. Espera a que termine su ejecución (puede comprobarlo mediante `condor_q`) y comprueba que el sistema le ha devuelto los resultados y el log correctamente.
Resultado	Satisfactorio. El programa se ha ejecutado y ha devuelto automáticamente los resultados al directorio señalado.

Tabla 12: Prueba someter localmente en Linux, universo Java

Título	**Someter remotamente en Linux**
Requisitos	*Central Manager* con función *Submit* funcionando, una o más máquinas *Execute*. Un usuario con cuenta en el *Central manager* y certificado/clave SSL.
Objetivo	Someter un trabajo desde una máquina remota (cualquiera menos el *Central Manager*) y ver qué ocurre con el trabajo.
Desarrollo	El usuario se conecta a cualquier máquina *Execute* del *pool*, ejecuta el comando `condor_submit –remote scheduler@ (nombre_fich.sub)`. Espera a que termine su ejecución (puede comprobarlo mediante `condor_q –name scheduler@`), y entonces ejecuta el comando para recuperar los resultados: `condor_transfer_data –name scheduler@ user` , y para borrar los trabajos completados de la cola: `condor_rm –name scheduler@ -constraint 'JobStatus =!= 2'`
Resultado	Satisfactorio. El usuario ha podido mandar el trabajo, recuperarlo y borrarlo después.

Tabla 13: Prueba someter remotamente en Linux

Título	**Flocking**
Requisitos	Dos máquinas *Central Manager* con los servicios *Submit* y *Execute*. Una de ellas con Linux y otra con Windows. También es posible delegar los servicios a otras máquinas, pero que tengan el mismo sistema operativo.
Objetivo	Probar a someter trabajos de Linux en Windows y viceversa, para ver que, al no haber ninguna máquina disponible en el propio *pool*, éstos migran a otro donde sí que hay.
Desarrollo	Se ha dispuesto de dos máquinas *Central Manager*, la del *pool* de Amazon (CentOS), y la del laboratorio del I3A (Windows). Se han creado dos ficheros ejecutables, uno para Windows y otro para Linux, desde un fichero fuente de prueba. El de Windows, se someterá desde la máquina de Amazon, con un requisito `OpSys == "WINDOWS"`, mientras que el de Linux, se someterá desde el ordenador del I3A.
Resultado	Los trabajos, al no haber ninguna máquina disponible en su *pool* que cumpla los requisitos, migran a otro donde sí que la hay, y se pueden ejecutar.

Tabla 14: Prueba *flocking*

Título	**Prueba en condiciones reales**
Requisitos	6 instancias de Amazon EC2, una por cada laboratorio del Ada Byron, más el *Central Manager*. Todas ellas con el mismo rol y la misma configuración que queremos implementar en las máquinas reales.
Objetivo	Comprobar el correcto funcionamiento de todo el sistema en su conjunto, con la configuración definitiva.
Desarrollo	Iniciamos primero la instancia del *Central Manager*, y después todas las demás. Comprobamos con *condor_status* que se reconocen todas las máquinas. Sometemos varios trabajos desde el *Central Manager* (en local) y desde una de las otras máquinas (en remoto), para comprobar acto seguido la cola con *condor_q*, con varios usuarios.
Resultado	Correcto. Los trabajos se distribuyen entre todas las máquinas disponibles (siempre que se cumplan los requisitos). Al acabar la ejecución, los trabajos sometidos en local se borran de la cola y se mandan los resultados al directorio desde el que se sometieron. Los sometidos en remoto se quedan en la cola a la espera de ser recogidos.

Tabla 15: Prueba en condiciones reales

La prueba en condiciones reales es la que nos permitirá determinar que el sistema en su conjunto funciona realmente como habíamos planteado inicialmente.

V.3.2. Casos de fallo

Aquí vamos a poner a prueba el sistema, los posibles fallos que se pueden dar, y cómo reacciona ante ellos. Es el apartado de pruebas más largo, pues hay muchas formas en las que el sistema puede fallar. Las siguientes tablas muestran las pruebas de fallos que hemos realizado.

Título	**Caída del *Central Manager***
Requisitos	*Central Manager* funcionando, al menos una máquina *Execute* que ejecute un trabajo sometido por otra máquina *Submit*, y uno o más trabajos en cola.
Objetivo	Comprobar que los trabajos en ejecución terminan, y los que están en cola, permanecen en ella hasta que vuelve a estar disponible el *Central Manager*.
Desarrollo	Se dispone de un *Central Manager*, una máquina *Submit* y otra *Execute*, todas ellas independientes entre sí. Se someten varios trabajos desde el *Submit*, de los cuales al menos uno se empieza a ejecutar. En ese momento, apagamos el *Central Manager*, bien con `condor_off`, bien con `poweroff`. Vemos la cola de trabajos desde la máquina *Submit*, mediante `condor_q`.
Resultado	Satisfactorio. Los trabajos que esperan en la cola se quedan esperando, mientras que los que acaban, devuelven sus resultados. Pueden someterse nuevos trabajos, aunque no sean enlazados.

Tabla 16: Prueba caída del *Central Manager*

Título	**Caída del *scheduler***
Requisitos	*Central Manager* con función *Submit* funcionando. Una o más máquinas *Execute* disponibles.
Objetivo	Comprobar que los trabajos sometidos terminan de ejecutarse, pero no se mandan de vuelta los resultados, y podrían llegar incluso a perderse si se demora mucho su puesta en marcha de nuevo.
Desarrollo	Sometemos un trabajo en local, y antes de que termine su ejecución, forzamos la caída del demonio *condor_schedd*, matando el proceso correspondiente. Comprobamos el estado del *pool* (`condor_status`), de la cola de trabajos (`condor_q`) y hacemos un `ls` para ver que no llegan los ficheros de resultados.
Resultado	Satisfactorio. Los trabajos se quedan a la espera de que el *scheduler* vuelva a estar activo para poder mandar los resultados pertinentes.

Tabla 17: Prueba de caída del *scheduler*

Título	**Caída conjunta de *Central Manager* + *scheduler***
Requisitos	*Central Manager* con función *Submit* funcionando. Una o más máquinas *Execute* disponibles.
Objetivo	Comprobar que los trabajos sometidos terminan de ejecutarse, pero no se mandan de vuelta los resultados, y podrían llegar a perderse si se demora mucho su puesta en marcha de nuevo.
Desarrollo	Sometemos un trabajo en local, y antes de que termine su ejecución, apagamos la máquina *Central Manager*, matando el proceso correspondiente. Comprobamos el estado del *pool* (`condor_status`), de la cola de trabajos (`condor_q`) y hacemos un `ls` para ver que no llegan los ficheros de resultados.
Resultado	Satisfactorio. Los trabajos se quedan a la espera de que el *scheduler* vuelva a estar activo para poder mandar los resultados pertinentes.

Tabla 18: Prueba caída conjunta de *Central Manager* + *scheduler*

Título	**Un *Execute* deja de estar disponible (pasa al estado Owner)**
Requisitos	*Central Manager* funcionando, una máquina *Execute* libre, ejecutando algún trabajo con la posibilidad de disponer de ella físicamente.
Objetivo	Comprobar que los trabajos en ejecución se matan cuando alguien utiliza la máquina.
Desarrollo	Se ha utilizado el ordenador del laboratorio del I3A, por poder disponer físicamente de él como administrador. Se ha unido éste al *pool* de Amazon, se le ha enviado un trabajo para ejecutar, se ha dejado libre y luego, mientras ejecutaba, llega alguien y toca ratón o teclado. Comprobamos desde el *Central Manager* o cualquier máquina *Submit* que el estado del ordenador del laboratorio pasa de *"Unclaimed"* a *"Owner"*, y el trabajo que estaba ejecutando mata.
Resultado	Satisfactorio. El trabajo pasa de nuevo al estado *Idle* hasta que vuelve a ser re-enlazado por otra máquina disponible.

Tabla 19: Prueba *Execute* pasa al estado Owner

Título	**Apagado ordenado de un *Execute***
Requisitos	*Central Manager* funcionando, una máquina *Execute* ejecutando algún trabajo y alguna otra libre.
Objetivo	Comprobar que los trabajos en ejecución terminan y migran a otra máquina libre (con o sin *checkpoint*).
Desarrollo	Se ha realizado la prueba en máquinas virtuales de Amazon. Tomamos una de ellas cuando está ejecutando un trabajo y hacemos un *condor_off* (apagado ordenado). Comprobamos desde el *Central Manager* o cualquier máquina *Submit* que la máquina deja de aparecer en el *pool*, y el trabajo en ejecución migra a otra máquina libre.
Resultado	Satisfactorio. El trabajo en ejecución pasa inmediatamente al estado *Idle* hasta ser re-enlazado.

Tabla 20: Prueba apagado ordenado de un *Execute*

Título	**Apagado brusco de un *Execute***
Requisitos	*Central Manager* funcionando, una máquina *Execute* libre y ejecutando algún trabajo.
Objetivo	Comprobar que los trabajos en ejecución terminan y migran a otra máquina libre
Desarrollo	Se ha realizado la prueba en máquinas virtuales de Amazon. Tomamos una de ellas cuando está ejecutando un trabajo y matamos los procesos de HTCondor mediante *kill -9*. Comprobamos desde el *Central Manager* o cualquier máquina *Submit* que la máquina deja de aparecer en el *pool* (es posible que tarde un poco, debido a que las revisiones son periódicas), y el trabajo en ejecución migra a otra máquina libre.
Resultado	Satisfactorio. El trabajo en ejecución pasa al estado *Idle*, aunque no inmediatamente, ya que no se ha notificado al colector de la caída de la máquina.

Tabla 21: Prueba apagado brusco de un *Execute*

Título	**Apagado ordenado de un *scheduler* de alta disponibilidad**
Requisitos	*Central Manager* funcionando, dos máquinas *Submit* con la cola de trabajos compartida, y algún trabajo en ejecución y en cola.
Objetivo	Comprobar que con la caída de una máquina, los trabajos sometidos desde ella migran a otra disponible.
Desarrollo	Comprobamos el estado actual de la cola de trabajos. Apagamos una de las máquinas con el comando *poweroff*, o el servicio, con *service condor stop* o *condor_off*. Acto seguido, comprobamos la cola de trabajos (*condor_q*) desde otra máquina *Submit*. Para poder comprobar diferencias, deben tratarse de trabajos de largo tiempo de ejecución.
Resultado	El trabajo en ejecución termina y devuelve los resultados, pero no se admiten nuevos trabajos ni se puede consultar el estado del pool (`condor_status`)
NOTA	Este caso se ha probado en Amazon EC2, pero en el despliegue real nos ha sido imposible implementar la función de *alta disponibilidad*, de modo que este caso no se llegará a dar hasta que se implemente.

Tabla 22: Prueba apagado ordenado de un *scheduler* de alta disponibilidad

Gracias a este tipo de pruebas podemos hacernos a la idea de la magnitud del problema que supone cada fallo en el sistema.

V.3.3. Casos adicionales

Aquí describimos otras pruebas que debemos realizar, que al tratarse de casos especiales no podemos englobar en casos de funcionamiento normal o de fallo. Las tablas 23 y 24 muestran dichas pruebas.

Tanto en el caso de las máquinas con IP pública como con las de IP privada hemos utilizado instancias de Amazon EC2 configuradas anteriormente, creadas a través de una imagen de la máquina *Execute*. La diferencia con las demás es que esta vez, no las hemos introducido en la red VPC que simula la red de los laboratorios, sino que se han creado en una red aleatoria, elegida por Amazon. Como todas las máquinas tienen una IP privada y otra IP pública, y ambas apuntan a la misma máquina de forma única, para un caso hemos cogido la privada y para otro, la pública.

Título	**Añadir máquinas con IP pública**
Requisitos	*Central Manager* conocido y disponible con IP pública
Objetivo	Comprobar que el colector reconoce nuevas máquinas y las añade correctamente al *pool*.
Desarrollo	Se ha utilizado un ordenador personal de un laboratorio del I3A, con Windows 7. Se ha iniciado HTCondor con una configuración muy similar a las de las máquinas de Amazon EC2, y luego se ha comprobado que hay una comunicación bidireccional entre las dos partes, a través del comando *condor_status*.
Resultado	Satisfactorio. Desde ambas partes se reconoce el estado del *pool*.

Tabla 23: Prueba añadir máquinas con IP pública

Título	**Añadir máquinas con IP privada**
Requisitos	*Central Manager* conocido con IP pública, y servidor *CCB* activo. Permisos de lectura y escritura para las nuevas máquinas.
Objetivo	Comprobar que el *CCB* registra correctamente las nuevas máquinas privadas, y se establece una comunicación bidireccional entre ellas y el *Central Manager*.
Desarrollo	Esta prueba se ha realizado dos veces. Primero con una máquina virtual de CentOS de Amazon, no perteneciente a la misma red VPC que el *Central Manager*. Después, con un ordenador personal de casa, con IP privada y sistema operativo Windows 7. En ambas pruebas se ha configurado HTCondor de forma similar (cambiando los parámetros de red), introduciendo esta vez, las variables `CCB_ADDRESS` y `PRIVATE_NETWORK_NAME`.
Resultado	Satisfactorio. El *CCB* registra las máquinas y estas reconocen el *pool* y son accesibles para recibir trabajos.

Tabla 24: Prueba añadir máquinas con IP privada

Estas pruebas adicionales nos permitirán comprobar que el sistema es escalable, que es posible añadirle más recursos en cualquier momento para mejorar su productividad.

Anexo V – Manual de usuario

Aquí vamos a explicar lo básico para que un usuario pueda utilizar HTCondor [B10] sin problemas. Hay dos formas de acceder a HTCondor: de forma local, en el *Central Manager*, o de forma remota desde cualquier otra máquina con acceso al mismo.

En este anexo se pretende explicar únicamente las características básicas que debe saber un usuario para la utilización del sistema. El manual completo de HTCondor puede ser consultado en la página oficial [W40].

V.1. Acceso local

En este modo accedemos directamente al *Central Manager*, ya sea físicamente, por SSH o por escritorio remoto (costoso, no recomendado). Esta función se recomienda para usuarios especiales como el mediador de la infraestructura GIDHE [W2]. No así para usuarios comunes, con posibilidad de acceder a cualquier otra máquina del *pool*.

Para someter un trabajo localmente, únicamente es necesario que el usuario se conecte directamente a la máquina que en ese momento esté corriendo el *scheduler* (si hay espejos, puede que no siempre sea el mismo) y ejecutar `condor_submit <fichero.sub>`.

Una vez termine el trabajo, automáticamente devolverá los resultados en los ficheros especificados al someter el trabajo, así como los posibles logs. Acto seguido, se elimina automáticamente de la cola de trabajos.

V.2. Acceso remoto

En este modo no accedemos de manera directa al *Central Manager*, sino a cualquier otra máquina *Execute* de nuestro *pool*. Para un usuario normal, es preferible someter sus trabajos de esta forma, porque así evitamos la sobrecarga que supone el acceso concurrente de muchos usuarios a la misma máquina.

Para someter un trabajo de forma remota hay algunos cambios relevantes:

- El usuario deberá tener una cuenta en el *Central Manager* con el mismo nombre, UID y GID que desde la máquina que está sometiendo. Esto no impone ninguna restricción, ya que en entornos académicos es habitual que el sistema de *login* esté centralizado entre todos los recursos.
- Al ejecutar el comando **`condor_submit`**, además debe añadir el argumento **`-name scheduler@`**, que es el nombre del *scheduler* que le hemos puesto para evitar poner la dirección completa.
- Cuando el trabajo termine de ejecutarse, esta vez no se eliminará de la cola. Se quedará a la espera de que el usuario que lo sometió lo recoja. Para ello tendrá que ejecutar **`condor_transfer_data -name scheduler@ [user | <cluster> |`**

<cluster.proc>]. Las opciones que se añaden al final se refieren, respectivamente, a transferir todos los trabajos del usuario, los del grupo <cluster> o un proceso en concreto <cluster.proc>.

- Como los trabajos sometidos remotamente no se eliminan automáticamente de la cola, tendremos que eliminarlos a mano, una vez recogidos sus resultados. Para ello será necesario ejecutar `condor_rm –name scheduler@ [user | <cluster> | <cluster.proc> | -constraint <expr>]`. Para la opción más común, que sería borrar únicamente los trabajos que ya se han completado, habría que introducir: **`condor_rm –name scheduler@ -constraint 'JobStatus =!= 2'`**.

Anexo VI – Integración con la Infraestructura del GIDHE

Tal y como hemos mencionado anteriormente, uno de los objetivos de este proyecto es el de integrar nuestra plataforma HTCondor desplegada en el ámbito universitario, con una infraestructura de computación distribuida desarrollada previamente.

En este anexo describiremos dicho proceso de integración, comenzando por la visión general de la infraestructura, siguiendo por el diseño de un nuevo mediador y finalizando con la implementación del mismo.

VI.1. Descripción de la Infraestructura de computación del GIDHE

El grupo de investigación GIDHE [W2] ha desarrollado una infraestructura orientada a servicios para el despliegue y la ejecución de *workflows* científicos en entorno de computación heterogéneos. Esta infraestructura integra y gestiona de forma transparente un amplio conjunto de recursos de computación (recursos tipo clúster, *grid*, *cloud*, etc.), y los ofrece al usuario como un único y potente entorno de ejecución.

La figura 23 muestra el diseño arquitectural de la infraestructura el cual está formado por tres capas. En la parte superior, la capa de interfaz de usuario está compuesta por los entornos de programación de workflows que pueden ser usados para desarrollar las aplicaciones ejecutadas sobre la infraestructura (por ejemplo, Taverna [B23], Triana [B24], Kepler [B25], etc.).

La capa intermedia, llamada capa de ejecución, es la responsable de gestionar el despliegue de las aplicaciones. Los componentes que constituyen esta capa están integrados en torno al bus de mensajes compartido, inspirado en el modelo de coordinación Linda [B14], utilizando como mecanismo de comunicación asíncrona. Existen dos tipos de componentes: los componentes de gestión y los mediadores. Los primeros son responsables de proporcionar la funcionalidad necesaria para gestionar el ciclo de vida de las aplicaciones: meta-scheduling, tolerancia a fallos, movimiento de datos, etc. Por su parte, los mediadores facilitan el acceso a los recursos de computación integrados en la infraestructura. Cada mediador es capaz de interaccionar con una infraestructura de computación concreta, encapsulando sus detalles específicos, con el objetivo de enviar tareas para su ejecución, monitorizar el estado de las tareas, recuperar los resultados finales, etc.

Por último, la capa compuesta por las infraestructuras de computación integra los recursos de computación heterogéneos accesibles. En la actualidad se han integrado el clúster HERMES del Instituto de Investigación en Ingeniería de Aragón (I3A) [W4], los grid AraGrid [W3] y PireGrid [W41] del Instituto de Biocomputación y Física de Sistemas Complejos (BIFI) [W42], y la infraestructura *cloud* Amazon EC2 [W5].El usuario final accede a estas infraestructuras como un todo y no es consciente de los diferentes paradigmas de computación integrados (clúster, *grid* y *cloud*) o de las distintas tecnologías middleware utilizadas para gestionar estas infraestructuras (HERMES utiliza el middleware HTCondor [B10] y AraGrid y PireGrid utilizan gLite [B17]).

Más información sobre la infraestructura de computación del GIDHE puede consultarse en [B2] [B3] [B19] [B20] [B22].

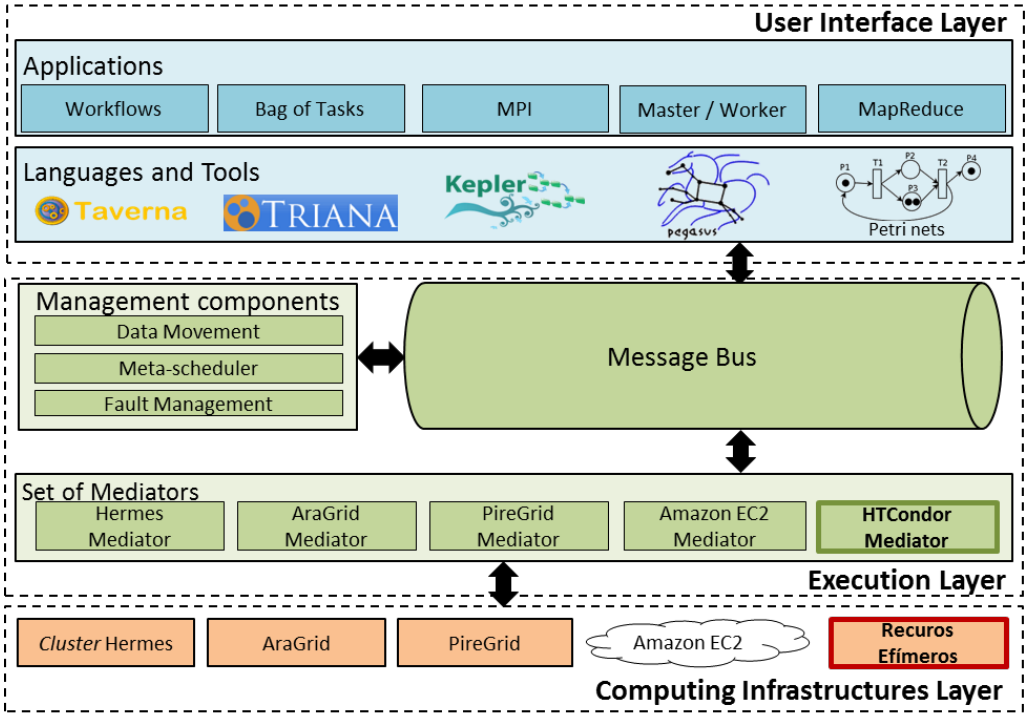


Figura 23: Arquitectura de la infraestructura

Una de las partes más importantes de la infraestructura, y la que más atañe a este proyecto, son los mediadores englobados en la capa de ejecución. El diseño de los mismos puede observarse en la figura 23. Este diseño es común para todos los mediadores, independientemente de la infraestructura y el middleware con el que interactúen, mientras que la implementación de los mismos varía según estos dos aspectos. En lo que respecta a los componentes que forman el mediador:

El **Job Manager** se encarga de gestionar y coordinar la ejecución de los diferentes trabajos enviados al mediador. Para ello interactúa con el **Bus** de mensajes para obtener nuevos trabajos y enviar sus resultados, gestiona el movimiento de los datos generados como resultado a la ubicación adecuada y mantiene una lista de los trabajos en ejecución junto con los identificadores asignados. El **Middleware Adapter** es el encargado de traducir la descripción del trabajo sometido, la cual es independiente del entorno de ejecución, en una descripción que pueda ser ejecutada por el middleware específico con el que interactúa. Asimismo, prepara los datos necesarios para poder ejecutar el trabajo. El **Middleware Executor** obtiene la descripción del trabajo a ejecutar y envía el mismo al middleware para que éste proceda a su ejecución. Finalmente, el **Job End Monitor** se encarga de detectar cuando un trabajo ha terminado y notificar al Job Manager.

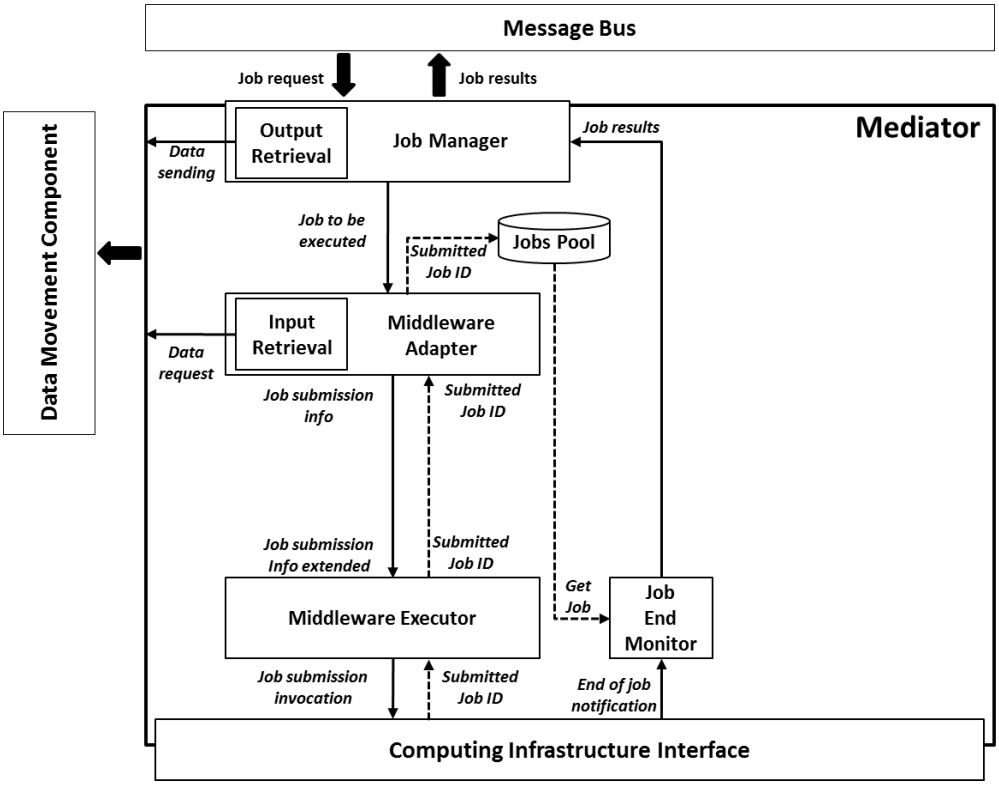


Figura 24: Esquema del mediador original

VI.2. Diseño de un mediador de recursos efímeros

En este proyecto se ha extendido la infraestructura presentada anteriormente para que sea capaz de integrar entornos de computación compuestos por recursos efímeros de ámbito académico. Más concretamente, se ha extendido el diseño de los mediadores de la infraestructura para que incluyan nuevas funcionalidades encaminadas a gestionar este tipo de entornos. La inclusión de estas nuevas características resuelve el problema de que los mediadores estaban diseñados para ser utilizados en entornos de computación estables en los que los recursos no pueden ser añadidos o eliminados de forma dinámica por causas externas y sin previo aviso (salvo aparición de errores). Por tanto, mediante el desarrollo de este nuevo mediador se consigue realizar un mejor aprovechamiento de los recursos integrados.

La figura 25 muestra el diseño extendido del mediador, remarcando los nuevos componentes añadidos al mismo, los cuales se detallan a continuación:

- ***Resource Occupation (Ocupación de Recursos)*:** Componente que recibe un fichero con información de la ocupación de los recursos, lo traduce y lo almacena en el Resource Registry. Debe ser capaz de gestionar la inclusión de cambios en la ocupación definida.

- ***Resource Monitor (Monitor de Recursos)*:** Componente que monitoriza de forma periódica el estado de los recursos (libre, ocupado, eliminado, etc.). Adicionalmente, detecta cuando un recurso es añadido o eliminado y lo notifica para que se guarde esa información.

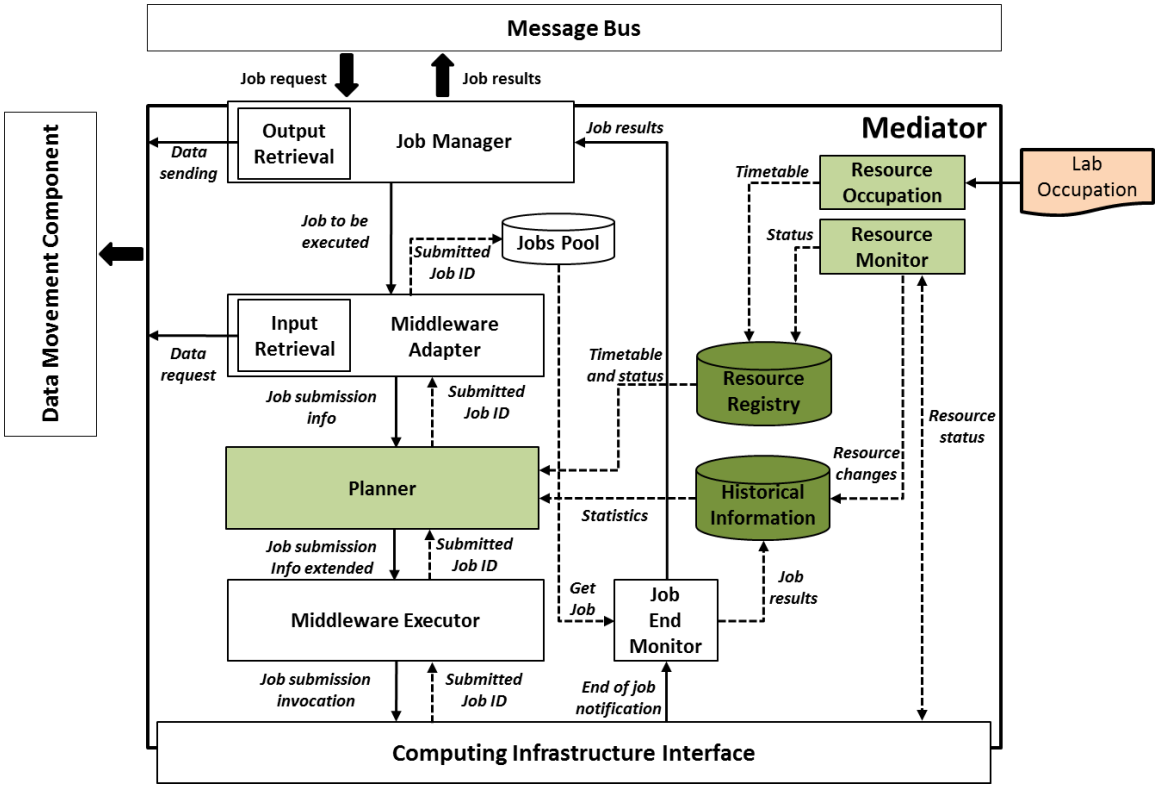


Figura 25: Esquema del mediador con funcionalidades añadidas

- ***Resource Registry (Registro de Recursos)*:** Almacén que contiene el estado de los recursos (procedente del Resource Monitor) e información acerca de su ocupación tanto pasada como futura (proporcionada por el Resource Ocuppation). Todos los datos deben ser almacenados de manera persistente en disco.

- ***Historical Information (Información Histórica)*:** Almacén de información que contiene información histórica sobre los trabajos ejecutados (procedente del Job End Monitor) y eventos referentes a la adición/eliminación de los recursos integrados en el entorno de computación (procedente del Resource Monitor). También permite obtener estadísticas sobre la utilización de los recursos y el rendimiento de los trabajos que puedan ser utilizadas durante la planificación. Todos los datos deben ser almacenados de manera persistente en disco.

- ***Planner (Planificador)*:** Componente que permite seleccionar el recurso o grupo de recursos concretos en los que se ejecuta un trabajo. Recibe la información del trabajo a someter y la extiende mediante un algoritmo de planificación o *scheduling* que determina dónde se ejecuta el trabajo. Para ello, puede utilizar diferentes fuentes de información como son: el estado actual de los recursos y la ocupación prevista para los mismos (obtenidos del Resource Registry) y/o estadísticas de uso, tanto de los propios recursos como de ejecuciones pasadas de la misma aplicación (proporcionadas por el Historical Information).

VI.3. Implementación de un mediador de recursos efímeros

En lo que respecta a la implementación del mediador, se ha tomado como base la implementación del mediador de HERMES que también es gestionado por el middleware HTCondor y se ha extendido con las funcionalidades y los componentes indicados en el apartado anterior.

VI.3.1. Gestión de los almacenes de información

Uno de los requisitos de los almacenes es que los datos sean almacenados de forma persistente en disco para que puedan ser utilizados entre diferentes ejecuciones y el mediador sea más tolerante a fallos. Para permitir esta funcionalidad hemos utilizado la librería MapDB [W49] de Java que nos proporciona estructura de datos de alto nivel para almacenar los datos y gestiona su uso tanto en disco como en memoria. MapDB gestiona de forma completamente transparente la escritura de los datos en disco, de forma que el programador sólo tiene que gestionar los contenidos en memoria. Además, MapDB también permite obtener los datos almacenados en disco de forma transparente cuando lanzamos nuevamente el mediador, liberando al programador de esta tarea.

Para la implementación de los almacenes, se ha creado un componente (clase Java) con el mismo nombre del almacén que es el encargado de interactuar con las estructuras de datos que forman cada almacén. De este modo, estas clases contienen todas las funciones necesarias para permitir tanto su utilización.

VI.3.2. Detalles de implementación de los componentes añadidos

En este apartado, incluimos algunos detalles de implementación de los componentes del mediador que consideramos relevantes.

- ***Resource Occupation (Ocupación de recursos)*:** Recibe un fichero de texto con un formato concreto, que contiene la información de la ocupación de los laboratorios del Departamento de Informática e Ingeniería de Sistemas [W1], la cual puede encontrarse en la página web del Departamento. Esa información es parseada y almacenada en el Resource Registry. Un fichero de ejemplo es el siguiente:

```
        L (24/03/14)     M (25/03/14)   F (26/03/14)  J (27/03/14)    V (28/03/14)
08..09  _____________    ___45a_______     Festivo     __345a5b__6b_   ___45a_______
09..10  1_3__________    1_345a_______     Festivo     __345a5b__6b_   1__45a_______
10..11  1_34________R    1_345a______R     Festivo     12345a5b__6bR   1__45a______R
11..12  1_34________R    1234________R     Festivo     12345a______R   1__45a5b____R
12..13  12__5a______R    123_5a_______     Festivo     123_5a5b_____   1_345a5b_____
13..14  12__5a______R    1_3_5a_______     Festivo     123_5a5b_____   1_345a5b_____
14..15  ____5a_______    ____5a_______     Festivo     ______5b_____   _____________
15..16  1__45a______R    12__5a_______     Festivo     ____5a_______   _____________R
16..17  1__45a______R    123_5a__6a___     Festivo     __3_5a_______   ___4________R
17..18  123_5a______R    _2345a__6a__R     Festivo     1_3_5a_______   ___45a______R
18..19  123_________R    1234__5b6a6bR     Festivo     1_345a5b_____   1__45a______R
19..20  1_3__________    1_34__5b6a6bR     Festivo     123_5a5b_____   1___5a_______
20..21  __3__________    ___4__5b6a6bR     Festivo     123_5a5b_____   _____________
```

Este fragmento representa una semana de clase. Cada columna es un día de la semana; cada fila, una hora; y cada celda indica qué laboratorios están ocupados durante esa hora. El formato usado para describir si un laboratorio está ocupado es incluir el identificador del laboratorio ("1" para lab001, "2" para lab002, y así hasta "R" para lab102) si está ocupado y utilizar el carácter '_' en la posición correspondiente a ese laboratorio si está libre. Cuando es festivo, simplemente se muestra la palabra "Festivo".

El fichero se procesa por bloques, con una estructura igual que el bloque de ejemplo de arriba. Para cada bloque, se obtiene primero la fecha de la primera línea y después, línea por línea se obtienen y se almacenan todos los laboratorios que aparecen como ocupados.

- ***Resource Monitor (Monitor de Recursos)*:** Monitoriza periódicamente el estado de los recursos disponibles. Para ello, ejecuta el comando `condor_status –xml` en el nodo principal de HTCondor de forma periódica y configurable. El comando anterior devuelve toda la información del estado de los recursos en formato xml y el componente extrae de la misma los datos más relevantes y los almacena en el Resource Registry.

- ***Resource Registry (Recurso de Registros)*:** Tiene dos estructuras de datos: un mapa hash para el horario de ocupación de los laboratorios, y una tabla hash para el estado de los recursos. En ambos casos la información se almacena jerarquizada para facilitar su utilización.

- ***Historical Information (Información histórica)*:** Tiene dos estructuras de datos: una tabla hash para almacenar los logs de los trabajos ejecutados y otra tabla hash para almacenar el listado de eventos de adición/eliminación de cada recurso. Además, el componente proporciona métodos para obtener estadísticas relevantes a partir de dichos logs (tiempo medio de ejecución de una aplicación, tiempo medio que un recurso está disponible y tiempo medio durante el cual un recurso no puede ser utilizado). En el futuro se plantea la inclusión de más estadísticas y la utilización de nuevas estructuras de datos para almacenar las mismas y no tener que calcularlas de forma completa.

- ***Planner (Planificador)*:** Recibe como parámetro de entrada la información para someter un trabajo a HTCondor, decide (mediante un algoritmo de planificación) en qué recursos se puede ejecutar el trabajo y lo añade a la información recibida. Es un intermediario entre el HTCondorAdapter y el HTCondorExecutor. Para el algoritmo de planificación sólo se usa la información de ocupación de los laboratorios (la utilización del resto excede los objetivos del proyecto y se plantea como trabajo futuro). Con esta información, el planificador añade datos sobre los laboratorios o recursos personales en los que se recomienda ejecutar un trabajo, de forma que, HTCondor intentará ejecutar el trabajo en los recursos indicados y sólo utilizará un recurso no recomendado en el caso de que los recursos recomendados estén todos ocupados. Para ello se ha utilizado la función `rank` de HTCondor.

Anexo VII – Gestión del proyecto

En esta sección se describe de forma global la planificación de este proyecto, así como sus fases, la metodología y el esfuerzo dedicado a su realización.

VII.1. Metodología de desarrollo

En este proyecto se ha seguido una metodología de desarrollo basado en el ciclo de vida en cascada, con realimentación. Cada fase del proyecto contiene las etapas de análisis, diseño, implementación y pruebas, como se muestra en la figura 26.

Se trata de un proceso lineal, aunque también es posible volver a una etapa anterior, al haber detectado un error o una alternativa mejor. De esta forma, el resultado que obtenemos es mejor (retroalimentación).

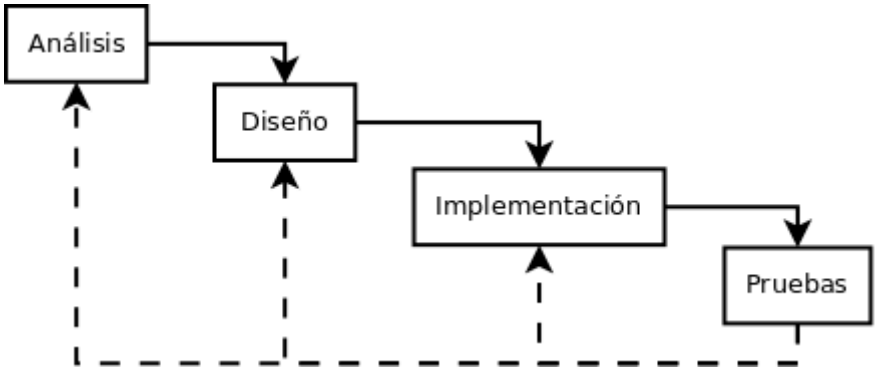


Figura 26: Ciclo de vida en cascada con retroalimentación

VII.2. Fases del proyecto

El proceso de desarrollo de este proyecto ha seguido una evolución que se puede dividir en varias fases de acuerdo con su naturaleza. A continuación se listan y detallan estos pasos y las etapas que los componen:

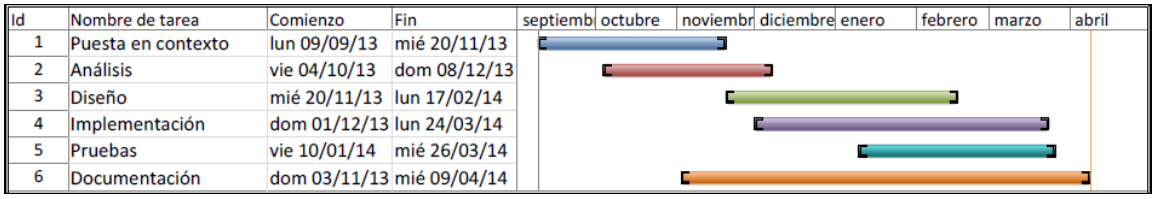
1. **Puesta en contexto.** La primera fase consistió en estudiar el contexto en el que se ubicaba el proyecto. En leer un gran número de artículos y buscar información sobre las distintas tecnologías de computación distribuida. A continuación se enumeran varias de ellas:
    - Computación clúster.
    - Computación *grid*.
    - Computación *cloud*.
2. **Análisis.** En esta fase se analizó toda la información obtenida anteriormente. Este análisis sirvió para elegir una alternativa entre todas las disponibles (HTCondor). Las tareas fueron:

- Análisis de los recursos disponibles (laboratorios del Departamento de Informática e Ingeniería de Sistemas)
- Análisis de las necesidades de los trabajos científicos a ejecutar.
- Análisis de las plataformas: sus ventajas, inconvenientes y posibilidad de adaptación.
- Definición de los requisitos del sistema.

3. **Diseño.** En esta fase se planteó el esquema general del sistema, y las funcionalidades que iba a tener el mismo.
    - Esquema de configuración de HTCondor.
    - Funcionalidades a implementar.
    - Comprobación de la viabilidad de las funcionalidades.
4. **Implementación.** Esta fase ha consistido en poner en práctica todo el diseño planteado anteriormente. Primero en un entorno virtual, posteriormente en uno real, y finalmente, la integración a la Infraestructura del GIDHE.
    - Instalación de HTCondor en el entorno virtual.
    - Configuración de HTCondor en el entorno virtual
    - Integración de HTCondor en la Infraestructura del GIDHE.
5. **Pruebas.** Esta fase se ha realizado conjuntamente con la implementación, y después de la misma. Su función ha sido la de comprobar que lo implementado funcionaba correctamente.
    - Planteamiento de las pruebas a realizar.
    - Ejecución de las pruebas.
    - Verificación de los resultados.
6. **Documentación.** Esta fase se ha realizado a lo largo de todo el proyecto. Su función ha sido la de documentar todo el análisis, así como justificar todas las decisiones de diseño e implementación, y los resultados.
    - Documentación de las fases de análisis y diseño.
    - Justificación de las decisiones de diseño e implementación
    - Estudio de limitaciones.
    - Planteamiento del trabajo futuro.
    - Creación de un manual de usuario.
    - Documentación del trabajo realizado.

VII.3. Gestión de tiempo y esfuerzo

En la figura 27 se muestra el diagrama Gantt con la distribución del tiempo dedicado a las tareas. Exceptuando la presentación pública.



Id	Nombre de tarea	Comienzo	Fin
1	Puesta en contexto	lun 09/09/13	mié 20/11/13
2	Análisis	vie 04/10/13	dom 08/12/13
3	Diseño	mié 20/11/13	lun 17/02/14
4	Implementación	dom 01/12/13	lun 24/03/14
5	Pruebas	vie 10/01/14	mié 26/03/14
6	Documentación	dom 03/11/13	mié 09/04/14

Figura 27: Diagrama de Gantt

En el diagrama se puede observar que excepto la fase de documentación, que comprende casi todo el proyecto, las demás son relativamente lineales, con alguna superposición de forma que se cumple el ciclo de vida en cascada que se comentó en el apartado anterior.

El proyecto comenzó a mediados de septiembre de 2013, justo después de la segunda convocatoria de exámenes, y la última edición se realizó el 9 de abril de 2014.

En la figura 28 se ve el porcentaje de tiempo de cada tarea

Figura 28: Diagrama de distribución de tiempo

El tiempo **total** invertido en este proyecto asciende a **602 horas**. De las cuales, 110 pertenecen a la fase de puesta en contexto, 79 a la de análisis, 94 a la de diseño, 115 a la de implementación y despliegue, 62 a la de pruebas y 142 para la documentación.

VII.4. Supervisión del proyecto

Durante el desarrollo de este proyecto, se han llevado a cabo una serie de reuniones con los directores del proyecto de forma periódica. La distribución en el tiempo no es completamente

regular, por existir fases del desarrollo que requerían de menos supervisión. El esquema general de las reuniones es el que sigue:

1. Evaluación del trabajo realizado desde la última reunión.
2. Discusión o debate de ideas, dudas e impresiones.
3. Planificación de nuevos objetivos o metas a medio/corto plazo, así como a largo plazo.
4. Planificación de la siguiente reunión.

VII.5. Herramientas de gestión utilizadas

Para la correcta realización de este proyecto, consideramos importante trabajar de manera descentralizada. Es decir, que tanto la documentación como los demás recursos generados (ficheros de configuración, figuras, material en general) pudieran ser accedidos desde cualquier lugar. De esta forma se podía trabajar desde cualquier ordenador, sin la limitación que supone tener que trabajar siempre desde el mismo sitio.

En un primer lugar se planteó la idea de utilizar un sistema Wiki como por ejemplo DokuWiki [W48]. Pero en vez de eso al final elegimos trabajar desde Dropbox [W44] con todo lo necesario, como si fuera una carpeta compartida entre el alumno y los directores del proyecto.

La herramienta Dropbox nos proporciona una ventaja importante, y es que el propio programa implementa un servicio de versiones para los documentos, que con cualquier error, podemos recuperar una versión más antigua.

El software utilizado para la realización del proyecto es el siguiente:

- **Dropbox** [W44]. Para el almacenamiento descentralizado de todos los recursos generados.
- **Microsoft Word 2010** [W45]. Para la edición de la memoria.
- **Microsoft Excel 2010** [W45]. Para el recuento del número de horas dedicadas al proyecto.
- **NetBeans 7.4** [W46]. Para la creación del mediador de la Infraestructura del GIDHE, en Java.
- **Google Calendar** [W47]. Para la gestión de las citas compartidas.

Y los recursos hardware utilizados han sido:

- Ordenador sobremesa de trabajo, en el laboratorio del grupo GIDHE en el I3A.
- Ordenador portátil personal, en el domicilio propio.
- Ordenador ultra-portátil personal, en bibliotecas públicas.
- Ordenadores de los laboratorios del edificio Ada Byron.

Glosario

- **ClassAd:** Es un lenguaje descriptivo que proporciona una estructura flexible para, entre otras cosas, unir recursos con trabajos.
- **Demonio (Linux):** Programa que se ejecuta en segundo plano (no interactivo). En Windows se llama **servicio**.
- **Espejo:** Es un servidor o máquina que contiene una réplica exacta de otro. Estas réplicas u espejos se suelen crear para facilitar descargas grandes y facilitar el acceso a la información aun cuando haya fallos en el servicio del servidor principal.
- **IaaS,** Infraestructura como Servicio: en este modelo se subcontrata el equipamiento necesario (hardware, almacenamiento, servidores y componentes de red) para soportar sus aplicaciones. Todo ello sobre Internet.
- **Máquina virtual:** Es un software que simula a una computadora y puede ejecutar programas como si fuese una computadora real.
- **Middleware:** Es un software intermedio que facilita la homogeneidad de un sistema heterogéneo a vista del usuario.
- **PaaS,** Plataforma como Servicio: los servicios que se entregan son sistemas operativos y servicios asociados sobre Internet sin descargas o necesidad de instalación.
- **Pool:** Conjunto de ordenadores, similar a un clúster pero en computación *grid*.
- **SaaS,** Software como Servicio: en este modelo las aplicaciones son alojadas por un proveedor de servicios y están disponibles a sus clientes sobre una red, normalmente Internet.
- **SSI:** Single System Image. Es un clúster de máquinas que aparecen como una sola. Oculta la naturaleza heterogénea y distribuida de los recursos, y los presenta a los usuarios y a las aplicaciones como un recurso computacional unificado y sencillo.
- **VPN:** Virtual Private Network. Es una tecnología de red que permite una extensión segura de la red local (LAN) sobre una red pública o no controlada como Internet.
- **Workflow:** Consiste en el estudio de aspectos operacionales de una actividad de trabajo, esto es, cómo se realizan y estructuran las tareas, cuál es su orden correlativo, cómo se sincronizan, cómo fluye la información y cómo se hace su seguimiento. Generalmente se modelan a través de redes de Petri.

Bibliografía

Referencias bibliográficas

[B1] I. Taylor, E. Deelman, D. Gannon and M. Shiedls, *Workflows for e-Science*, 2006.

[B2] S. Hernández de Mesa, *Framework para el despliegue automático de workflows científicos en entornos Grid*, Universidad de Zaragoza, 2011.

[B3] J. Fabra, S. Hernández, J. Ezpeleta, P. Álvarez, "*Solving the interoperability problem by means of a bus. An experience on the integration of grid, cluster and cloud infrastructures*", in Journal of Grid Computing, Special Issue on Interoperability, Federation, Frameworks and Application Programming Interfaces for IaaS Clouds, pp. 1-25 2013

[B4] N. Sadashiv, S.M Dilip Kumar, *Cluster, Grid and Cloud Computing: A Detailed Comparison*, 6th International Conference on Computer Science & Education (ICCSE), 2011, IEEE, pp. 477-482.

[B5] I. Couto Vivas, *Sistema de computación masiva en Sun Grid*, Universitat Politècnica de Catalunya, 2009.

[B6] F. Magoules, J. Pan, *Introduction to Grid Computing*, CRC Press, 2009.

[B7] I. Foster, C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2003)

[B8] L. M. Vaquero, L. Rodero-Merino, J. Caceres, M. Lindner, *A break in the clouds: towards a cloud definition*. ACM SIGCOMM Computer Communication Review, 39(1), 50-55, 2008

[B9] I. Foster, Y. Zhao, I. Raicu, S. Lu, *Cloud computing and grid computing 360-degree compared*, Proceedings of the Grid Computing Environments Workshop, GCE 2008, pp. 1–10, 2008

[B10] D. Thain, T. Tannenbaum, M. Livny, *Distributed Computing in Practice: The Condor Experience*, University of Wisconsin-Madison, 2004.

[B11] P. Sempolinski, D. Thain, *A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus*, University of Notre Dame, 2010.

[B12] J. Herrera Sanz, *Modelo de programación para infraestructuras Grid computacionales*, Capítulo 2, Universidad Complutense de Madrid, 2008.

[B13] A. Marosi, Z. Balaton, P. Kacsuk, D. Drótos: *SZTAKI desktop Grid: adapting clusters for desktop Grids*, Davoli, F., Meyer, N., Pugliese, R., Zappatore, S. (eds.) Remote Instrumentation and Virtual Laboratories, pp. 133–144. Springer, New York (2010)

[B14] J.L. Vázquez-Poletti, E. Huedo, R. Santiago, I. Martín, *Una visión global de la tecnología Grid*, Centro de Astrobiología (CSIC-INTA), Universidad Complutense de Madrid, 2004.

[B15] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke, *Condor-G: A Computation Management Agent for Multi-Institutional Grids*, University of Wisconsin, 2002.

[B16] T. Kosar, M. Livny, *Stork: Making Data Placement a First Class Citizen in the Grid*, University of Wisconsin-Madison, 2004.

[B17] E. Laure, *Programming the Grid with gLite. In: Computational Methods in Science and Technology*, 2006.

[B18] E. Martí Garacía, *Plug-in Globus para WINGS*, Capítulos 2 y 3, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 2009.

[B19] D. Gelernter: *Generative communication in Linda*. ACM Transactions on Programming Languages and Systems 7, pp. 80–121, 1985.

[B20] N. Carriero and D. Gelernter, *Linda in context*. Communications of the ACM, Vol.32, Num.4, pp. 444-458, 1989.

[B21] I. Taylor, E. Deelman, D. Gannon, M. Shiedls, *Workflows for e-Science*, 2006.

[B22] J. Fabra, P. Álvarez, J.A. Bañares, J. Ezpeleta, *RLinda: A Petri Net Based Implementation of the Linda Coordination Paradigm for Web Services Interactions*,

[B23] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M.R. Pocock, P. Li, T. Oinn, *Taverna: a tool for building and running workflows of services*, Nucleic acids research 34, 2006

[B24] I. Taylor, M. Shields, I. Wang, A. Harrison, *The Triana workflow environment: Architecture and applications*, Workflows for e-Science (pp. 320-339), Springer London, 2007

[B25] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E.A. Lee, J. Tao, Y. Zhao, *Scientific workflow management and the Kepler system. Concurrency and Computation*, Practice and Experience 18, 2006

[B26] J.L. Vázquez-Poletti, *Ejecución eficiente de flujos de trabajo computacionales en entornos Grid*, Universidad Complutense de Madrid, 2009.

[B27] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov: *The Eucalyptus open-source cloud-computing system*. Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID ’09, pp. 124–131 (2009).

Referencias web

[W1] DIIS: http://diis.unizar.es/

[W2] GIDHE: https://i3a.unizar.es/es/content/gidhe

[W3] AraGrid: http://www.aragrid.es/

[W4] Hermes: https://i3a.unizar.es/es/content/cluster-de-supercomputaci%C3%B3n-hermes

[W5] Amazon EC2: http://aws.amazon.com/es/ec2/

[W6] openMosix: http://openmosix.sourceforge.net/

[W7] PBS: http://www.pbsworks.com/

[W8] OpenPBS: http://www.mcs.anl.gov/research/projects/openpbs/

[W9] LSF: http://www-03.ibm.com/systems/services/platformcomputing/lsf.html

[W10] Grid Engine: http://www.univa.com/products/grid-engine

[W11] GridCafe, *grid computing*: http://www.gridcafe.org/EN/grid-in-30-sec.html

[W12] HTCondor: http://research.cs.wisc.edu/htcondor/

[W13] BOINC: http://boinc.berkeley.edu/

[W14] SZTAKI: http://szdg.lpds.sztaki.hu/szdg/

[W15] XtremWeb-CH: http://www.lsds-rg.org/xtremwebch/

[W16] What is Cloud Computing?: http://thegadgetsquare.com/1552/what-is-cloud-computing/

[W17] Amazon AWS: http://aws.amazon.com/es/

[W18] Windows Azure: http://azure.microsoft.com/es-es/

[W19] Google Compute Engine: https://cloud.google.com/products/compute-engine/

[W20] Eucalyptus: https://www.eucalyptus.com/

[W21] OpenNebula: http://opennebula.org/

[W22] Nimbus: http://www.nimbusproject.org/

[W23] Métodos de seguridad: https://htcondor-wiki.cs.wisc.edu/index.cgi/tktview?tn=589,56

[W24] OpenSSL: https://www.openssl.org/

[W25] Kerberos: http://web.mit.edu/kerberos/

[W26] Rackspace: http://www.rackspace.com/es/

[W27] Amazon VPC: http://aws.amazon.com/es/vpc/

[W28] Renew: http://www.renew.de/

[W29] Arquitectura de BOINC: http://chessbrain.net/LFBOF2005/images/boinc.gif

[W30] ARC: http://www.nordugrid.org/arc/

[W31] UNICORE: http://www.unicore.eu/

[W32] EMI: http://www.eu-emi.eu/

[W33] dCache: http://www.dcache.org

[W34] Historia de EMI: http://www.isgtw.org/feature/emi-user-survey-surprising-start

[W35] Globus Toolkit: http://toolkit.globus.org/toolkit/

[W36] GridCafe, Globus Toolkit: http://www.gridcafe.org/EN/globus-toolkit.html

[W37] http://research.cs.wisc.edu/htcondor/manual/current/3_8Checkpoint_Server.html

[W38] http://research.cs.wisc.edu/htcondor/manual/v8.0/3_11High_Availability.html#sec:HA-negotiator

[W39] https://htcondor-wiki.cs.wisc.edu/index.cgi/wiki?p=HowToMixFirewallsAndHtCondor

[W40] HTCondor manual: http://research.cs.wisc.edu/htcondor/manual/current/index.html

[W41] PireGrid: http://www.piregrid.eu/

[W42] BIFI: http://bifi.es/en/

[W43] Amazon S3: http://aws.amazon.com/es/s3/

[W44] Dropbox: https://www.dropbox.com/

[W45] Microsoft Office: http://office.microsoft.com/es-es/

[W46] NetBeans: https://netbeans.org/

[W47] Google Calendar: https://www.google.com/calendar

[W48] DokuWiki: https://www.dokuwiki.org/dokuwiki

[W49] MapDB: http://www.mapdb.org/